Large Vocabulary Recognition of On-line Handwritten Cursive Words

by

Giovanni Seni

A dissertation submitted to
the Department of Computer Science of
the State University of New York at Buffalo
for the degree of
Doctor of Philosophy

August, 1995

©Copyright by Giovanni Seni
All Rights Reserved

Large Vocabulary Recognition of On-Line Handwritten Cursive Words

by Giovanni Seni

Abstract

A critical feature of any computer system is its interface with the user. This has led to the development of user interface technologies such as mouse, touchscreen and pen-based input devices. Since handwriting is one of the most familiar communication media, pen-based interfaces combined with automatic handwriting recognition offers a very easy and natural input method. Pen-based interfaces are also essential in mobile computing because they are scalable. Recent advances in pen-based hardware and wireless communication have been influential factors in the renewed interest in on-line recognition systems.

On-line handwriting recognition is fundamentally a pattern classification task; the objective is to take an input pattern, the handwritten signal collected on-line via a digitizing device, and classify it as one of a pre-specified set of words (i.e., the system's lexicon). Because exact recognition is very difficult, a lexicon is used to constrain the recognition output to a known vocabulary. Lexicon size affects recognition performance because the larger the lexicon, the larger the number of words that can be confused. Most of the research efforts in this area have been devoted to the recognition of isolated characters, or run-on hand-printed words. A smaller number of recognition systems have been devised for cursive words, a difficult task due to the presence of the letter segmentation problem (partitioning the word into letters), and large variation at the letter level. Most existing systems restrict the working dictionary sizes to less than a few thousand words.

This research focused on the problem of cursive word recognition. In particular, I investigated the issues of how to efficiently deal with large lexicon sizes, the role of dynamic information over traditional feature-analysis models in the recognition process, the incorporation of letter context and avoidance of error-prone segmentation of the script by means of an integrated segmentation and recognition approach, and the use of domain information in the postprocessing stage. These ideas were used to good effect in a recognition system that I developed; this system, operating on a 21,000-word lexicon , was able to correctly recognize 88.1% (top-10) and 98.6% (top-10) of the writer-independent and writer-dependent test set words respectively.

ACKNOWLEDGMENT

My deepest thanks go to my family, especially my wife Ana, for her love and support while I have been working on this dissertation.

I am deeply grateful to my thesis advisor Rohini K. Srihari. She has funded, encouraged, and educated me since I chose my thesis topic. I look forward to continuing our friendship in the years to come.

I would like to express my appreciation to the other members of my thesis committee, Nasser M. Nasrabadi and Sargur N. Srihari. Professor Nasrabadi introduced me to the theory of neural networks and inspired my interest in this topic. Professor Srihari first provided me the opportunity to work at CEDAR where I was exposed to the science of recognition, analysis and interpretation of digital documents.

There were times when I wondered if I would ever finish this thesis. Thanks for the encouragement of my friends, especially Dar-Shyang Lee, Jenchyou Lii, and Jian Zhou, who motivated me and gave me constructive criticisms. Special thanks to Keith Bettinger who kindly helped proofread this manuscript and who introduced me to the art of X-windows programming. In addition, Ajay Shekhawat has provided much technical assistance in performing my thesis experiments.

I would like to thank the people I shared an office with: Kripa Sundar, Stayvis Ng, and Bobby Kleinberg. They all greatly contributed to my work.

My thanks also go to the many subjects who provided handwriting for my experiments, and to Dan Mechanic, Eric Wang, and Shu-Fang Wu who assisted on the truthing effort.

The CEDAR research group has been an excellent place to work because of the comradeship of its members, the sharing of knowledge and expertise, and its fine facilities. While my thanks go out to the group as a whole, I am particularly grateful to Ed Cohen who introduced me to scientific writing, and Evie Kleinberg with whom I have inspiring conversations.

To my parents and grandparents

Contents

1 Introduction **1**

1.1 Strategies for Cursive Word Recognition 6

1.2 Cursive Handwriting as a Temporal Signal 9

1.3 Research Issues 11

1.4 Outline of the Dissertation 12

2 Previous Work **13**

2.1 Segmentation-based Recognition 13

2.2 Whole-word Recognition 16

2.3 Psychology-related Research 23

2.4 Neural Network Approaches 24

2.5 Integrated Segmentation and Recognition 27

3 System Overview **31**

4 Preprocessing Module **35**

5 Filtering Module **40**

5.1 Syntactic Methods in Pattern Recognition 41

5.1.1 Formal Grammars and Recognition of Languages 42

5.2 The Task of the Filtering Module 44

5.3 Selection of Primitives 45

5.4 Generation of Matchable Words 49

5.5 Testing of Filtering Module 53

5.6 Discussion of Filtering Module 54

6 Recognition Module **56**

6.1 Artificial Neural Networks 57

6.1.1 The Backpropagation Algorithm 61

6.1.2 Feed-forward Networks and Pattern Recognition 63

6.1.3 The Time-Delay Neural Network 65

6.2 Trajectory Representation 67

6.2.1 Zone Encoding 71

6.2.2 Time Frames 72

6.2.3 Varying Duration and Scaling 72

6.3 Neural Network Recognizer 73

6.4 Neural Network Simulation 76

6.4.1 Training Signal 77

6.5 Output Trace Parsing 78

6.5.1 Missing Peaks 81

6.5.2 Delayed Strokes 82

6.6 String Matching 84

6.6.1 Extension of the Damerau-Levenshtein metric 87

6.7 Testing of Recognition Module 93

6.8 Discussion of Recognition Module 94

7 Conclusions **99**

A Production Rules for Syntactic Matching **102**

B A Typology of Recognizer Errors **105**

B.1 Refining the operations 106

B.2 The basic ordering 107

B.3 Additional constraints 109

B.4 Solving for the cost ranges 110

C Experimental Data **112**

C.1 Desirable Corpus Characteristics 112

C.2 The First25 Data Set 114

C.3 The Second25 Data Set 116

C.4 The Sentence Data Set 118

List of Figures

1 The *pen-based* interface. 2

2 The *on-line* and *off-line* word recognition problem. 3

3 Different handwriting styles. 4

4 Example of difficulties present in cursive word recognition. 5

5 Possible scheme for unconstrained handwriting recognition. 6

6 Illustration of the *segmentation-based* approach to cursive word recognition. 7

7 Illustration of the *word-based* approach to cursive word recognition. 8

8 *Static* vs. *Dynamic* representation of the handwriting signal. 10

9 Example of the use of y-minimas of the pen trace as possible segmentation points. 15

10 Example of no y-minima segmentation point. 16

11 Example of the classification procedure used by Earnest:1962. 17

12 Example of word-level feature vector used by Frishkopf and Harmon:1961. . . 19

13 Example of Freeman style coding scheme used by Farag:1979. 20

14 Example of word-level feature vector used by Brown and Ganapathy:1980. . . 22

15 The neural network scanning approach used by Martin:1992. 28

16 Overview of proposed system for large vocabulary recognition of on-line handwritten cursive words. 32

17 The Preprocessing module. 35

18 Example of noise present in on-line data. 36

19 Preprocessing example. 37

20 Preprocessing example. 39

21 The Filtering module. 40

22 Block diagram of a general syntactic pattern recognition system. 42

23 Examples of $downward$ strokes. 47

24 Examples of downward strokes in word images. 48

25 Examples of $retrograde$ pen motion in cursive characters. 49

26 The need for the rewrite rule $\alpha_i \rightarrow U; 1 \leq i \leq 3; \alpha_i \in \{A, D, B\}$. 49

27 Derivation of matchable words. 52

28 Filtering example. 53

29 The Recognition module. 57

30 Block diagram of a McCulloch-Pitts neuron. 59

31 A two-layer perceptron. 59

32 Schematic diagram of the back-propagation weight update rule. 63

33 A three-layer time-delay neural network (TDNN) used to recognize phonemes. 66

34 Schematic diagram of a hypothesized feed-forward network for letter identification. 67

35 Directional information used in the encoding of pen trajectory. 69

36 Example parameters used in the encoding of pen trajectory. 70

37 Zone encoding of the pen trajectory. 72

38 The architecture of a TDNN-style network for cursive word recognition. . . . 74

39 The procedure for generating target vectors for training patterns. 77

40 Output activation traces generated by the neural network recognizer. 79

41 The operation of the output trace parsing algorithm. 81

42 Detection of missing activation peaks. 83

43 Example of delayed-stroke processing. 85

44 The role of string matching in the Recognition module. 86

45 Examples of common "look-alikes" occurring in cursive handwriting. 91

46 Examples of weight kernels. 97

47 Words in the First25 data set. 116

48 Words in the Second25 data set. 117

49 Example of data truthing screen for cursive words. 120

50 The amount of data available in our handwriting corpus. 122

51 Test image examples. 123

List of Tables

1 Cost assignment for the refined set of operations. 91

2 The SUBSTITUTE table. 92

3 The SPLIT/MERGE table. 92

4 Valid PAIR-SUBSTITUTE possibilities. 93

5 Writer-dependent Test. 94

6 Writer-independent Test. 94

7 Refining the basic edit operations. 107

8 Cost assignment for the refined set of operations. 111

9 Variability factors covered by our handwriting corpus. 114

10 The First25 data set. 116

11 Common data pairs from a 21,000 word lexicon. 117

12 The Second25 data set. 118

13 Test data used for writer-independent evaluation of the Recognition module. 122

Chapter 1

Introduction

A critical feature of any computer system is its interface with the user. This has led to the development of user interface technologies such as mouse, touch-screen and pen-based input devices. They all offer significant flexibility and options for computer input; however, touch-screens and mice cannot take full advantage of human fine motor control, and their use is mostly restricted to data "selection" (i.e., as pointing devices). On the other hand, pen-based interfaces allow, in addition to the pointing capabilities, for other forms of input such as handwriting, gestures, and drawings. A pen-based interface consists of a transducer device and a fine-tipped stylus that is used to write directly on the transducer so that the movement of the stylus is captured (see Figure 1); such information is usually given as a time ordered sequence of x-y coordinates (digital ink). The most common of these transducer devices is the electronic tablet or digitizer, which typically has a resolution of 200 points/inch, a sampling rate of 100 points/sec, and an indication of "inking" (i.e., whether the pen is up or down).

Digital ink can often be passed on to recognition software that will convert the pen input into appropriate computer actions. The term on-line has been used to refer to

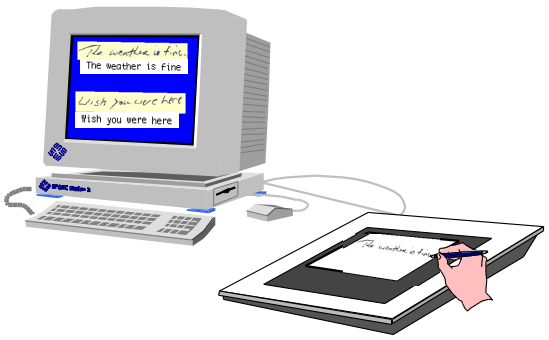


Figure 1: The *pen-based* interface. A digitizer generates x, y coordinates (digital "ink") when the pen is placed on or near it. Recognition software converts the ink into appropriate computer actions.

systems devised for the recognition of patterns generated with these type of devices as opposed to off-line techniques which take as input a static two-dimensional image representation instead (usually acquired by means of a scanner). Since handwriting is one of the most familiar communication media, pen-based interfaces combined with automatic handwriting recognition offers a very easy and natural input method. Furthermore, people tend to dislike pushing keys on a keyboard unless the task is routine data entry.

Recent hardware advances combining tablets and flat displays resulting in integrated input/output devices, and higher resolution and sampling rates, have been influential factors in the renewed interest in pen-based systems [95]. Pen-based interfaces are also essential in mobile computing (e.g., personal assistants and personal communicators) because they are scalable. Only small reductions in size can be made to keyboards before they become awkward to use; however, if they are not shrunk in size, they lose their portability. Handwriting recognition (the task of translating the image of a handwritten text into ASCII) is critical to the success of these devices if one is to be able to use them

with applications such as field data entry, note-pads, address books and appointment calendars.

On-line handwriting recognition is fundamentally a pattern classification task (see Figure 2); the objective is to take an input pattern, the handwritten signal collected on-line via a digitizing tablet, and classify it as one of a pre-specified set of words (i.e., the system's lexicon or reference dictionary). Because exact recognition is very difficult, a lexicon is used to constrain the recognition output to a known vocabulary. Lexicon size is an important factor conditioning recognition performance because the larger the lexicon, the larger the number of words that can be confused.

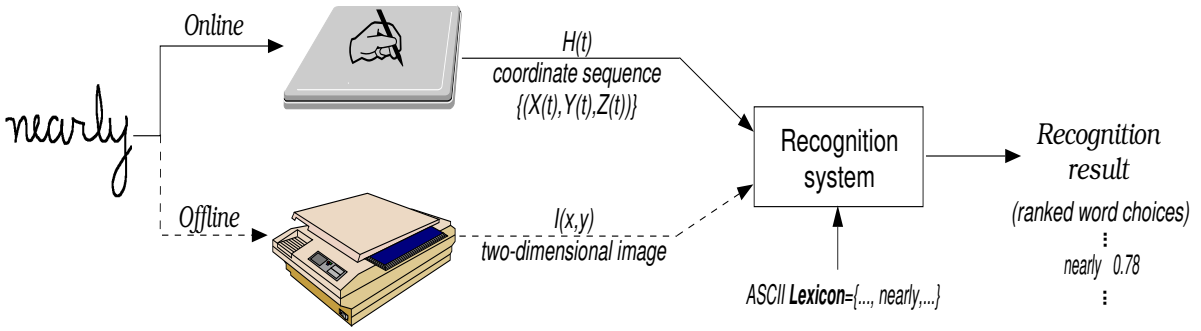


Figure 2: The *on-line* and *off-line* word recognition problem. Given a word image and a lexicon containing the word, the objective is to classify the image as one of the words in the lexicon. In the on-line case, handwriting is represented as a sequence of coordinates $H(t)$; in the off-line case, handwriting is denoted by a bitmap image $I(x, y)$.

Most of the research efforts in on-line handwriting recognition have been devoted to the recognition of isolated characters (particularly important for large-alphabet languages such as Chinese, with over 3000 different ideographs) [34, 62, 48, 71], or run-on hand-printed words [81, 28, 29] (see Figure 3); a significantly smaller number of recognition

systems have been devised for cursive words [82, 69, 23]. Many existing systems restrict the working lexicon sizes to less than a few thousand words; others have writer-dependent recognition capabilities only (i.e., they only recognize the writing of a single author).

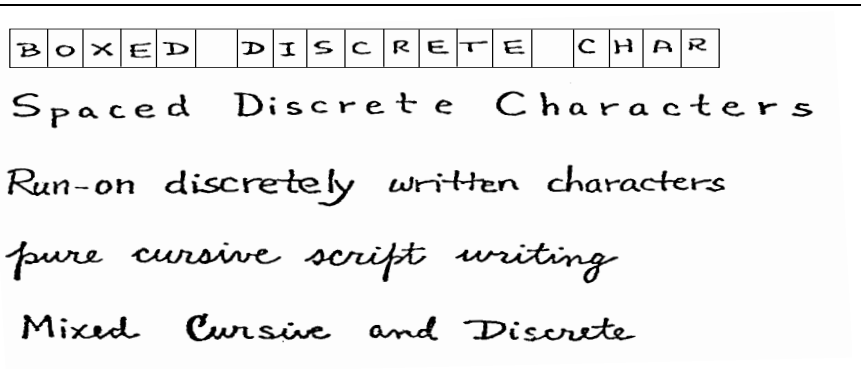


Figure 3: Different handwriting styles ordered, from top to bottom, according to the presumed difficulty in recognition (adapted from Tappert:1984).

Recognition of cursive handwriting is a difficult task mainly due to the presence of the letter segmentation problem (partitioning the word into letters), and large variation at the letter level (see Figure 4). Segmentation is complex because it is often possible to break up letters into parts that are in turn meaningful (e.g., the cursive letter 'd' can be subdivided into letters 'c' and 'l'). Variability in letter shape is mostly due to co-articulation (the influence of one letter on another), and the presence of ligatures which frequently give rise to unintended ("spurious") letters being detected in the script.

Other important applications of pen-based interfaces include recognition of Pitman's shorthand [57, 76], sketches and drawings [51] and signature verification [101, 75, 13].

In this thesis we focus on the problem of cursive word recognition using a large vocab-

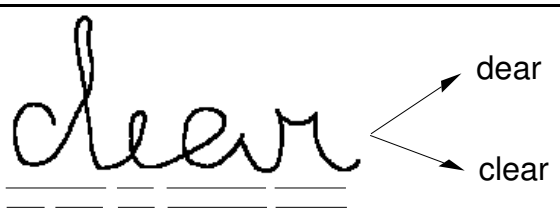


Figure 4: Example of difficulties present in cursive word recognition: segmentation of the script into letters is ambiguous, and ligatures often give rise to spurious letters (adapted from Edelman:1990).

ulary. A solution to the more general problem of recognizing unconstrained handwritten words (i.e., words that are written using a combination of cursive, discrete and/or run-on discrete styles) can be obtained once specialized algorithms have been developed to handle each basic writing style. Indeed, there is psychological evidence in support of separate processing systems used by humans for the recognition of typed and handwritten letters [10]. Individual algorithms could be combined by means of a word-style discriminator which first determines the writing style of the input word (or word fragment), and then applies the corresponding algorithm. A practical implementation of this idea was accomplished by Favata [21] in his work on off-line word recognition (see Figure 5). Another approach was recently suggested by Lee [56] who proposed the Dynamic Selection Network in his work for digit recognition; a multi-layer perceptron trained to take an image as input and output a number for each classifier being combined indicating how much confidence should be placed in the classifier's decision on the given image.

Finally, a word is in order about two important related problems: word boundary identification and linguistic or contextual post-processing. The former refers to the task of separating a line of handwritten text into words [85]; this is usually a required step

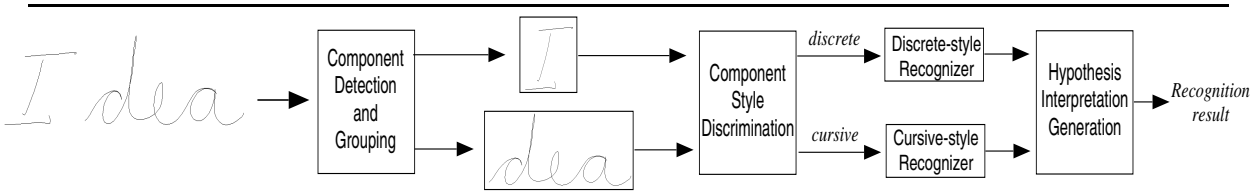


Figure 5: Possible scheme for unconstrained handwriting recognition. Specialized algorithms are used for the recognition of the individual components present in the input image according to the writing style (adapted from Favata:1992).

before word recognition algorithms can be used. The latter refers to the use of high-level contextual information, e.g. syntax, by means of applied language models [91] to detect and correct errors in the word recognition output. Both problems need to be address in order to develop systems for general text recognition. They, however, are out of the scope of this thesis.

1.1 Strategies for Cursive Word Recognition

Two major approaches have traditionally been used in cursive handwriting recognition: segmentation-based and word-based (also referred to as "holistic"). In the segmentation-based approach (see Figure 6), proposed initially by Mermelstein and Eden [66], each word is segmented into its component letters and a recognition technique is then used to identify each letter. Unfortunately, the nature of cursive script is such that the letter segmentation points (i.e., points where one letter ends and the succeeding one begins) can only be correctly identified when the correct letter sequence is known. On the other hand, recognition of characters can only be done successfully when the segmentation is

correct [23]. A relaxed segmentation criteria is commonly used whereby a large number of potential segmentation points are generated; this in turn can result in combinatorial complexity when combining multiple decisions about individual characters. Therefore, a recognition engine that performs character recognition and segmentation in parallel is desirable. Segmentation-based systems also make poor use of "contextual" information provided by neighboring characters.

Figure 6: Illustration of the *segmentation-based* approach to cursive word recognition. A "segmenter" generates candidate segmentation points, which can potentially represent character boundaries.

In the word-based approach [26, 15], rather than recognizing individual letters, a global feature vector is extracted from the input word (see Figure 7) and matched against a stored dictionary of prototype words; a distance measure is used to choose the best candidate. This word recognition method has the advantage of speed, and avoids problems associated with segmentation [38]. This recognition method reflects the human reading process more, which is not character by character but rather by words or even phrases [8]. The main disadvantages of this method are the need to train the machine with samples of each word in the established dictionary and the difficulty in devising word-level feature vectors that uniquely characterize words, thereby constraining vocabulary size.

It is believed that humans recognize words by executing a sequence of hypothesis

Ascenders	Descenders	NumLetters
1	1	3-5

Figure 7: Illustration of the *word-based* approach to cursive word recognition. A feature vector that describes each word as a whole is used; here, word 'why' is described by its number of ascenders, number of descenders, and estimated number of characters.

formation and comparison with some mentally stored image representation; the precise form of this representation remains unknown [66]. However, it is known that humans perform the more general task of object recognition following a 'coarse to fine' approach where decisions are based jointly on both large elements, and on smaller local details of the patterns. It is therefore natural to consider the above approaches to cursive word recognition to be complementary rather than mutually exclusive. Tentatively recognized words may be checked by making letter-analytical tests, while tentatively recognized letters can be tested to see whether they form words [26]. A goal of this research was to suggest an integrated segmentation and recognition model inspired by this argument which would constitute an intermediate position between the above fundamentally different approaches.

1.2 Cursive Handwriting as a Temporal Signal

A parallel has traditionally been drawn between cursive handwritten word recognition and continuous speech recognition. Both problems involve the processing of noisy lan-

guage symbol strings with ambiguous boundaries and considerable variations in symbol appearance. In addition to this initial similarity, the process of handwriting and continuous speech both generate signals that possess an inherent temporal structure. While temporal information of handwriting gets lost in the off-line case where handwriting is considered as a purely spatial domain function $H(x, y)$, it is available in the on-line case where handwriting is regarded as a time signal $H(t)$.

Recognition in the off-line case is more difficult because it is necessary to deal with accidental intersections present in the script (i.e., overlapping or touching characters) which result from a sloppy writer not moving the hand fast enough from left to right. Similarly, during the writing process, the pen can unintentionally separate from the paper causing letter elements, present in the ideal letter patterns, to be absent in the written script. These stroke absences and superfluous intersections significantly alter the topological pattern of the word, but have little or no influence on the "dynamic pattern" of the word (see Figure 8). It is therefore natural to hypothesize that the dynamic pattern of motion in cursive handwriting carries valuable information for recognition and less variability than the static geometric representation (assuming words are written naturally, e.g. not backwards). The time-based representation can be considered a source of misinformation as well. For instance, the letter 'E' can be written using multiple pen trajectories generating temporal variations that are not apparent in its static representation. While such variations in trajectory can be relatively large in isolated characters, the number of variations is limited when the word is written cursively (i.e., the pen trajectory is very

consistent).

Figure 8: *Static* vs. *Dynamic* representation of the handwriting signal. Superfluous intersections in the static representation of the script, (a), have little or no influence on the dynamic representation of it (b).

Recently, Time-Delay Neural Networks (TDNNs), a connectionist architecture developed for speech recognition, have been shown to be successful in learning to recognize time-varying signals. They outperformed HMMs (Hidden Markov Models) in a phoneme recognition task [98, 53]. Neural networks provide an effective approach for a broad spectrum of applications. In particular, they have been proven to be very competitive with classical pattern recognition methods, especially for problems requiring complex decision boundaries [42]. Moreover, because neural networks have automatic learning capabilities, they offer the potential of eliminating much of the hand-tweaking and lengthy development times associated with traditional recognition technologies [61]. It is a goal of this research to implement a TDNN-style recognition scheme based on cursive handwriting generation; the neural network-based recognizer will take low-level information about pen trajectory as input rather than feature vectors from a static 2-D image.

1.3 Research Issues

The problem investigated in this thesis is that of writer-independent large-vocabulary recognition of on-line handwritten cursive words. In particular, this research is concerned with the following issues:

1. *Lexicon reduction* : we want to formulate a filtering technique suitable for reducing a large lexicon (i.e, more than 20,000 words) to a smaller number of matchable words, which could then be passed to a more elaborated recognition algorithm for further processing. A reduced lexicon will limit the amount of work required during the string matching – postprocessing – stage (see below).

   The technique must be computationally efficient (i.e., very fast) and exhibit a degree of robustness and flexibility in responding to real-world data.

2. *Temporal representation*: we want to employ a representation scheme that preserves the inherent temporal structure of cursive handwriting and allow us to use the Time-Delay Neural Network (TDNN) architecture.

3. *Integrated segmentation and recognition*: we want to avoid an explicit segmentation procedure and to incorporate some form of "contextual" information in the recognition stage.

4. *String matching*: we want to develop a string similarity function that will allow us to effectively match the output of the neural network-based recognizer with the set

of matchable words.

The string matching function will be effective if it is capable of compensating for the type of errors present in the script recognition domain domain (e.g., characters are often "merged").

1.4 Outline of the Dissertation

We begin this thesis with a review of some prior approaches to on-line cursive handwriting recognition and some related techniques (Chapter 2). Later chapters present research carried out to develop a complete computer system that demonstrates solutions to the items enumerated above. In Chapter 3, an overview of the recognition system that has been been implemented is presented. Chapters 4, 5 and 6 describe the workings of individual parts of that system, including the preprocessing algorithms, the filtering technique responsible for reducing the lexicon, and the neural network-based technique used for character recognition. Chapters 5 and 6 include an experimental results section as well as a discussion section where possible extensions of the presented techniques are suggested. The dissertation is concluded with a summary of the contribution in Chapter 7.

Chapter 2

Previous Work

An attempt is made to synthesize the most salient features of some previously reported approaches to the on-line handwriting recognition problem. First, the traditional *segmentation-based* and *word-based* approaches are reviewed. Then, some relevant research done in neuro-psychology is discussed. Finally, work done in neural networks related to this problem is presented. Whenever possible, we give performance evaluation in terms of data sets used and lexicon sizes. It should be noted, however, that a direct comparison between these systems is not possible for various reasons: (i) some are intended for words, while others concern letters, (ii) recognition rates were not obtained with the same database or under similar conditions, (iii) time constraints are not always available, etc.

2.1 Segmentation-based Recognition

Segmentation based systems can be classified according to the type of features used to define the letter segmentation points of the script. One class of systems uses local maxima and minima in the x and/or y directions as possible segmentation points. A

second class of systems bases their segmentation techniques on results from psychophysical studies in cursive script production, to identify stroke (i.e., portions of the script between two consecutive segmentation points) boundaries by locating velocity troughs or curvature peaks. A third class of systems attempts to characterize the building elements of cursive script (e.g., letters and ligatures) and uses this information to locate the correct segmentation points. Systems based on segmentation can also be classified according to the techniques used to compare the extracted strokes from the script and stored prototypes. There are four main techniques that predominate for template matching [23]: elastic matching, Freeman coding, feature matching, and rule-based matching.

One of the earliest segmentation based system was developed by Mermelstein and Eden [66]. They used y-maxima and y-minima to segment words into a set of *up-strokes* and *down-strokes* ordered in time. Strokes were recognized by the statistical likelihood of belonging to twelve preselected classes, and the resulting ordered sequences of stroke categories were analyzed for possible mappings into a letter sequence that was a member of the output vocabulary of the system. Experiments were performed using 100 words (repeated samples of 12 different words) written by 4 subjects. Recognition accuracy ranged from about 90% to about 60%. In the former case the whole set of samples were used to compile stroke statistics, and subsequently the system was asked to recognize the same 100 samples. In the latter case, the machine was made to recognize the writing samples of subjects different from those on which the stroke statistics were based. Such deterioration in word recognition was an indication of the extent to which the stroke

description was subject dependent.

Ehrich and Koehler [17] segmented at all local y-minima ignoring those superfluous points associated with ornamental loops (e.g., the short down-stroke in the letter 'o'). Each down stroke ending on a y-minima, called pre-segment marks (PS), was initially classified according to the regions (defined by the positions of the base and half reference lines) in which its endpoints fall (see Figure 9).

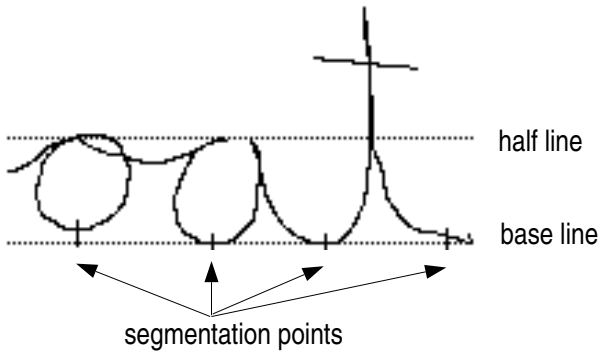


Figure 9: Example of the use of y-minimas of the pen trace as possible segmentation points as used by Ehrich and Koehler:1975.

Using the classification results of every pair of consecutive down-strokes, preliminary *substitution sets* were built. A substitution set is a set of characters that are the best alternatives for a given letter position inside a word. These sets were further refined by making use of geometric invariances and 15 feature measurements made on the data in the vicinity of each PS point. Experiments on a 300-word dictionary of seven letter words, prepared by three different writers, resulted in a 1.3% reject rate when the training and test data were identical, 18% when only half the training set was written by the writer of the test words, and 29.4% when the training set did not include samples by the writer of

the test words. Error rates for these experiments were very small, since when rejections occurred, no further attempts at classification were made.

Three major problems have been identified with the use of maxima and minima as segmentation points [23]: first, many minimas that do not represent segmentation points can occur and additional work is required to remove them. Second, minor variations in the letter style can add or delete maxima or minima. Third, there is sometimes no y-minima between letters and therefore no segmentation point is detected (see Figure 10).

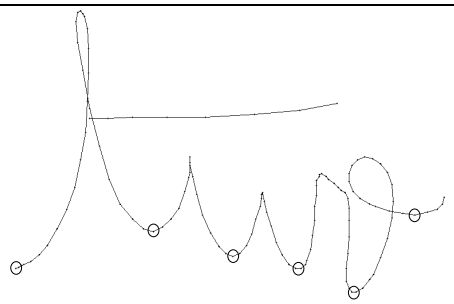


Figure 10: Example of no y-minima segmentation point: there is no y-minima between the letters 'w' and 'o'.

2.2 Whole-word Recognition

Earnest [15] developed a system for single word samples, written "more or less" horizontally and without capital letters. Using a 10,000-word dictionary (representing those words that occur more often), he approached script recognition as a problem of properly categorizing each script sample using a 7-bit feature vector (whether any *crossbars* were

found, number of *high* strokes, number of *low* strokes), and then performing a succession of discriminative tests to yield a progressively shorter word list (see Figure 11). To test the system, five subjects were asked to write 107 randomly selected words. The system correctly listed 65 (60% success). The resultant list contained 9 words on average to about 20 words in the worst case. This represents a discrimination ratio of 500 to 1 over the dictionary as a whole.

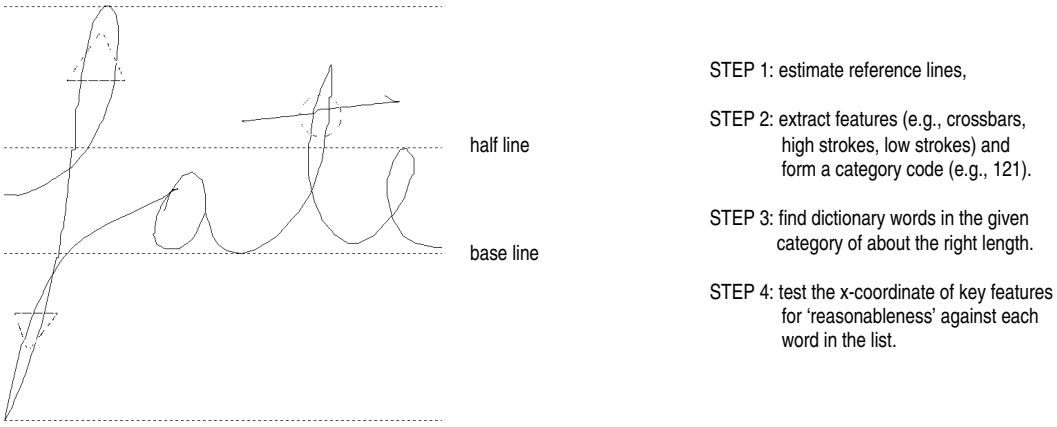


Figure 11: Example of the classification procedure used by Earnest:1962: (a) extracted features, and (b) major processing steps.

Frishkopf and Harmon [26] were also interested in finding a word representation scheme which could permit discrimination among a large vocabulary. Words were represented by an ordered list of *extreme* points (i.e., points at which either X_i or Y_i passes through a local maximum or minimum). Each extreme is associated with a 6-bit word that describes the presence or absence of the following properties (see Figure 12):

- Extreme type: X or Y,

- Extreme sub-type: *right* or *left* for X extremes; *upper* or *lower* for Y extremes,
- Slope: does the segment $(extreme_{i-1}, extreme_i)$ have *positive* or *negative* slope ?,
- Concavity: is the arc $(extreme_i, extreme_{i+1})$ *convex* or *concave* ?,
- Vertical extension: to which of the three amplitude groups (large *lower* extensions, large *upper* extensions, extremes of *intermediate* extent) does this Y extreme belong ? The vertical extent decisions are based on the relative amplitudes of all Y extremes within a word instead of reference lines. This property requires 2 bits but does not apply to X extremes.

The recognition process consists of a correlation comparison of the extreme listing of a test word, with the extreme listing of each dictionary word. Only those dictionary words which satisfy a length criterion (given in terms of number of extremes in the test word) are considered as candidates for identification. To avoid isolated coincidences, non-zero correlation is assigned only if two or more consecutive entry pairs are identical. Longer sequences of consecutive matching pairs are also given higher scores. After this comparison is carried out, one listing is *displaced* relative to the other by up to p positions, and the same procedure is performed again. This displacing mechanism allows to pick up coherent parts of the word when two samples of the same word contain different number of extremes. The sum of correlations at displacements $0, \pm 1, ..., \pm p$ yields a number which measures the similarity between the test word and a particular dictionary word.

To test the performance of the system, 5 people were asked to write a hundred word

Extreme #	Concavity	Slope	Ext. Type	Ext. Subtype	Vert.Extension
1	0	0	1	0	01
2	0	0	0	1	00
3	0	0	1	1	10
4	1	0	1	0	01
5	0	1	1	1	01
6	0	0	0	1	00
7	1	1	0	0	00
8	0	0	1	0	01
9	1	1	0	1	00
10	0	0	1	1	01
11	1	1	0	0	00
12	0	0	1	0	01
13	0	0	1	1	10
14	1	1	0	0	00
15	0	0	1	0	01
16	1	0	1	1	01
17	0	1	1	0	00
18	1	0	0	0	00
19	0	1	1	1	01
20	1	0	0	1	00
21	0	1	1	0	01
22	1	1	1	1	01
23	1	1	1	0	01

Figure 12: Example of word-level feature vector used by Frishkopf and Harmon:1961.

dictionary and 7 test sentences, comprising 32 words. For each test word, after ranking every dictionary entry which met the length criteria according to its correlation sum, the correct word was found among the top 2,5,10,20 words in 46%,54%,67%, and 85% of all cases, respectively. 11% of the test words ranked below 20th, and 4% were excluded on the basis of failing the length criteria. A disadvantage of the system, signaled by the authors, was the inability to make certain distinctions (e.g., 'clear' vs. 'dear'), primarily due to the lack of a metric in the extreme representation.

More recently, Farag [20] developed a system to recognize a small vocabulary of keywords, based on a Freeman style coding of the script and a Markov chain model to calculate a weighting when comparing the sample with a template word (see Figure 13). Hidden Markov models [77] (HMM) are a popular stochastic modeling technique. The states of the Markov chain correspond to the eight directional vectors (strokes) of the coding scheme. Each allowed word was represented as a collection of transition matrices M_j, each matrix corresponding to a particular time interval. An entry m_{pq} in the stochastic matrix M_j denotes the probability of a stroke q at time j having obtained a stroke p at time $j-1$; $0 \leq p, q \leq 7$. Since the number of strokes representing each word in the dictionary may vary from one word to the next, the last part of longer words were truncated to allow for a uniform handling during classification.

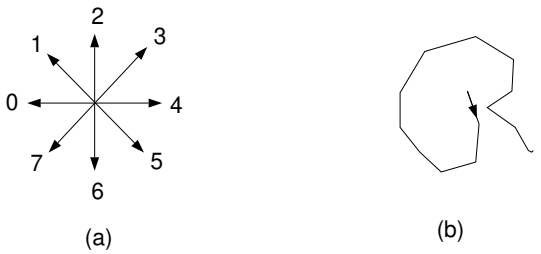


Figure 13: Example of Freeman style coding scheme used by Farag:1979: (a) coding directions, and (b) the representation of a letter R with code *6601123456755*.

A maximum-likelihood classifier scheme was used to select word w_j from the dictionary with the largest joint probability $P(z, w_j) = P(z \mid w_j) \cdot P(w_j)$ where $P(z \mid w_j)$ were calculated by selecting the appropriate entries from the matrices M_j and multiplying all these probabilities together. Using a testing set of 200 samples (20 versions of 10 different words written by ten authors) and a first-order Markov model, trained on the same set

of examples, the recognition rate was 98%. Using a second order Markov model, the result was 100% recognition. Farag concludes his report by indicating that his technique is appropriate for applications concerned with a limited vocabulary.

Brown and Ganapathy [8] developed a system with no constraints placed on character size, word length, writing speed, or character style, representing more relaxed conditions than in the previous systems. They used a set of features including the following:

- Y *maxima* and *minima*,
- *Dots* on the character 'i' and 'j',
- *Crossbars* of the characters 't' and 'x',
- *Cusps*, which are defined as rapid changes in stroke direction,
- *Retrograde strokes*, defined as strokes "flowing" from right to left,
- *Closures* (e.g., as in the character 'a'),
- *Direction of openings*, for those characters without the closure property (e.g., the character 'c'),
- *Threshold crossings* (i.e., crosses with the reference lines) used to determine *upper* threshold crossings (ascenders), *lower* threshold crossings (descenders), and *center* threshold crossings (word length).

The location of each feature occurrence in the script sample is specified using two set of *windows* that roughly divide the word into a number of regions equal to the estimated number of characters. The number of characters is approximated by dividing the number of central threshold crossings by empirically determined constant 2.65 (see Figure 14). The actual X or Y coordinates of the feature locations were discarded.

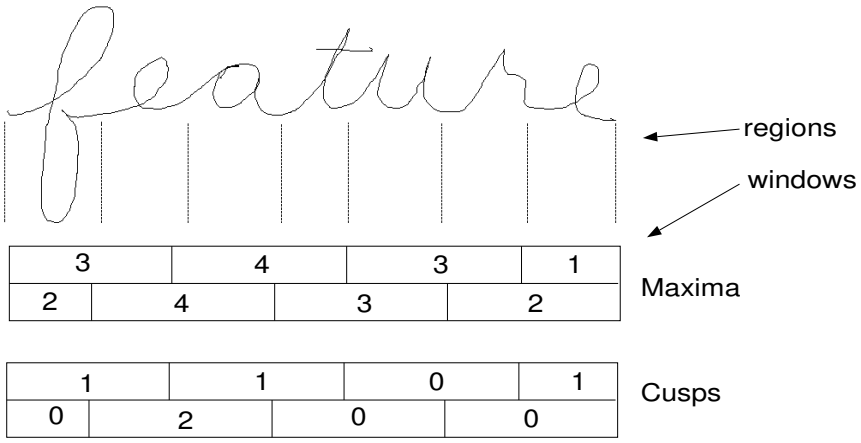


Figure 14: Partial feature vector for the word 'feature' as defined by Brown and Ganapathy:1980. Only the entries corresponding to the *maxima* and *cups* properties are shown.

Recognition was accomplished using a 3-nearest neighbor rule in which the word class having the largest number of samples (out of the 3) nearest to the unknown in the feature space is selected. Performance was evaluated using 10 samples of 22 randomly chosen words from three persons. Recognition rates ranged from 64.1% to 96.8% depending on which subset was used for training and which one was used for testing.

2.3 Psychology-related Research

I am not the first one to claim that in order to attain accuracy levels in cursive word recognition closer to those already achieved by optical character recognition (OCR) systems, the recognition technique should view cursive handwriting not as a two-dimensional image but rather as a continuous sequence of movements produced by a human hand. A psychological study by Zimmer [102] on the role of dynamic information in handwriting recognition, suggested that the most expeditious mental representation of handwriting is one that involves knowledge of the production method. In another experiment by Freyd [25], evidence was presented in support of the claim that the reader's tacit knowledge of the writing process (i.e., information about how letters are formed) facilitated recognition of distorted characters in static forms. Babcock *et al.* [2] carried out another experiment that further confirmed that readers are able to extract the underlying dynamic pattern of motion used to produce handwritten characters from the static traces. All these suggest that the recognition scheme should emphasize the use of dynamic or production information over static structural features.

Numerous models have also been proposed in the past aimed at understanding the bio-mechanical or neuro-psychological aspects of the human writing system (for a review see [74]). Some models are more oriented toward handwriting analysis, others toward handwriting generation (e.g., the coupled oscillator model of Hollerbach [40]). Models have also been classified as continuous or piecemeal [39] depending on whether they postulate the existence of basic strokes that are joined together to generate handwriting.

For example, Morasso *et al.* [70] developed a model where strokes (described by curved segments of given length, tilt angle and angular change) are used to reconstruct handwriting with the constraint that each stroke is generated with a symmetrical bell-shaped velocity profile, centered at a specified instant of time. Similarly, Maarse *et al.* [60] suggested that the control of the muscles involved in producing the writing movements is of a ballistic nature. Ballistic movements are extremely rapid actions that, once initiated, cannot be modified; they typically last less than a fraction of a second, so that feedback corrections are largely ineffective because reaction times are too long. Maarse's ballistic strokes have thus only a single velocity maximum and a typical duration.

2.4 Neural Network Approaches

Some of the models initially developed from a neuro-psychological point of view, were used in the design of feature extraction modules for recognition applications. Morasso *et al.* [69, 68] developed a system for writer-dependent cursive word recognition based on Kohonen's self-organized maps (SOMs) [50]. Words were segmented into strokes, via detection of points of minimum speed, which were coded as a nine-dimensional feature vector derived from a five-point polygonal approximation to the stroke. The resulting map after training became a "similarity map" where the distance between two units was proportional to the dissimilarity of strokes to which the different units responded. During recognition the sequence of coded strokes was scanned with six k-stroke maps ($k : 2 \rightarrow 7$; each map was intended to classify a k-stroke letter) producing a number of

ranked character matches that were subsequently passed to a lexical analyzer for filtering out non-valid words. A nearly 70% word recognition rate with a 4,000-word dictionary was achieved.

A similar stroke-based approach was adopted by Schomaker [82]. He segmented words into *kinematical strokes* (i.e., pieces of the word bounded by minima in the tangential pen-tip velocity) which were represented with fourteen features. Quantization of stroke shapes was accomplished by means of a single Kohonen network whose output units were labeled with possible stroke interpretations of the form $Name(I/N)$ (e.g., the label $a(1,3)$ means the first stroke in a three stroke letter 'a'). Using the "best match" only during recognition yielded a 50% correct word recognition rate (user specific). Allowing up to three multiple stroke interpretations increased recognition rate close to 90%.

Flann *et al.* [22] also segmented words into strokes using points of zero vertical velocity. Each stroke was represented by eight equally spaced points together with some approximations of the angular velocity and angular acceleration values at these points. During recognition six k-stroke input ($k : 1 \rightarrow 6$) multi-layer perceptrons were used instead of SOMs to scan the sequence of coded strokes. Contextual information was provided to the networks by means of the two adjacent strokes (i.e., a k-stroke network really received $(k+2)$ strokes as input). A word recognition rate of about 90% correct is reported for a writer-specific task using a 1,000-word dictionary during word interpretation.

Hakim *et al.* [35] avoided any form of segmentation. A bank of six recurrent neural

networks was developed, each trained to recognize a specific character, whose input consisted of the $x(t)$ and $y(t)$ signals only (the original coordinate sequence was slightly modified so as to make $x(t)$ and $y(t)$ to be stationary and bounded) which was fed sequentially to the networks. A *reconstruction* algorithm was subsequently used to build a list of character interpretations from the output sequences generated by the networks. A letter recognition rate of 84% was reported but the experiment was limited to the letters 'a', 'e', 'l', 'n', 'p', and 's'.

Hoffman and Skrzypek [39, 89] also took a "continuous" approach to cursive character recognition by avoiding segmentation into single strokes. A cluster of three-layer feed-forward neural networks that shared a common set of input nodes and whose output was collected by an independent *judge* layer was built. The horizontal axes of the X and Y velocity traces were normalized to values between 0 and 1. From these traces, the magnitude (and relative position) of each "major" positive and negative peak was extracted and fed as input to the network cluster together with the position of the zero vertical velocity crossings. A letter recognition rate close to 80% was reported on characters generated from off-line data using a line-following algorithm.

Guyon *et al.* [33, 34] used a TDNN style network for the recognition of digits and block capital letters. Although this work was not intended for cursive script, it is illustrative to see how they used time information instead of the raw 2D image representation. Characters were resampled to have 81 points, including pen-up points. Resampling is a preprocessing operation intended to make on-line data to be equal-spaced instead of

equal-timed usually by means of linear interpolation. Each point was substituted by a seven-component feature vector which encoded information about the direction and curvature, normalized coordinates and state of the pen (up/down) at that point. The sequence of 81 feature vectors (or *frames*) served as input to a 5-layer network which achieved classification accuracy of 96% after training on a database of over 12,000 samples. An analysis of the few errors revealed the system to be unable to recognize characters written with an unusual sequence of strokes ("even though the static pixel map does not look atypical"). As I mentioned in the Introduction, this argument could be raised against the use of temporal information in the recognition process. Letters can be written using multiple pen trajectories generating temporal variations that are not apparent in the static representation. However, while such variations could be relatively large for some isolated characters, it would seem that there are not many different ways of writing the letters inside cursive words, which is the focus of this research.

2.5 Integrated Segmentation and Recognition

Conventional segmentation-based algorithms for handwritten text recognition encounter difficulty if the characters are touching, broken or noisy. The difficulties arises from the fact that often one cannot properly segment a character until it is recognized yet one cannot properly recognize a character until it is segmented [47]. Some neural network models that simultaneously segment and recognize in an integrated system are now presented.

Martin *et al.* [63, 64, 73] developed a scheme called *centered-object integrated segmentation and recognition* (COISR), that simultaneously segments and recognizes ZIP Codes. The approach uses a *sliding window* concept where a neural network-based recognizer is trained to recognize what is centered in its input window as it slides along a digit field. A similar approach was used for speech synthesis in NETTalk [84] and in speech recognition [53].

The network uses 2D image input, with the input image tall enough to see one line of text, and wide enough to see several digits (see Figure 15). The architecture sequentially scans the input image, using a sliding window with a step size of 3 pixels, to create a possible segmentation at each scan point. The network is trained to both identify when its input *window* is centered over a character, and if it is, to classify the character.

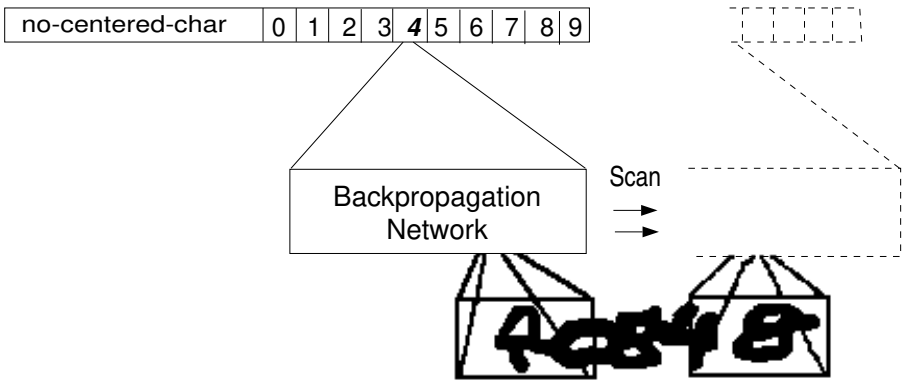


Figure 15: The neural network scanning approach used by Martin et all.,1992.

The output layer contains one unit per character category, and one unit associated to the state when there is no centered character in the window. In order to determine the ASCII string corresponding to the input word, a postprocessor was used to analyze the

output trace looking for significant valleys in the activation values of the *no-centered-character* unit. When the activation value of this unit falls below a threshold, the system classifies the character by determining which output unit has the highest activation value for this position in the word. Trained on 20,000 digit words (2 to 6 characters long), written by 800 different individuals, and tested on a separate set of 5,000 digit words achieved a word accuracy of 99% with a reject rate of 4.8%, 11.1%, 19.1%, 23.4% and 35.7% for 2-character, 3-character, 4-character, 5-character, and 6-character words respectively.

Rumelhart [79] devised a system for recognizing on-line cursive handwriting where the input script is broken up into strokes (using points where the y-velocity equals zero), each one encoded with the following "dynamic" parameters:

- net motion in the y-direction,
- net motion in the x-direction,
- net motion of the pen halfway through the stroke,
- x-velocity at the end of the stroke,
- ratio of x-frequency to y-frequency (the underlying dynamic model assumed that the x and y velocities could be described as sinusoidal).

The input to the network consisted of a sequence of up to 60 strokes ("an average word consists of only 20 strokes"), ordered according to to their x-coordinates. That is, words were presented to the network as a whole. However, the network was taught to

recognize individual letters. The output of the network is an activation two-dimensional grid with entries $Out[l, t]$ corresponding to the network's confidence in recognizing letter l at location t in the input. A dynamic programming postprocessor was used to find the best fitting word from a given dictionary. Trained on a huge database of about 650,000 characters obtained from words written by 100 donors, the reported recognition performance on "a reasonably large group of writers" is about 80.0% and 95.0% top-1 and top-5 respectively using a 1,000-word dictionary.

Chapter 3

System Overview

This research adopts an intermediate position between the segmentation-based and word-based approaches to word recognition, and attempts to incorporate the following three concepts relating to the cognition of cursive handwriting. First, the perception of words by humans is a two step process: characteristic letters are found in the word image which are used to select candidate words; an attempt is then made to align these words with the input image [87]. Second, the dynamic pattern of motion in cursive handwriting is generally consistent and carries valuable information for recognition [102, 2, 74]. Third, separating a character from its background is not a necessary preprocessing step for identifying the character. Accordingly, we use first a filtering technique that extracts a structural description for a given input word and uses it to quickly reduce a large lexicon (i.e, more than 20,000 words) to a more manageable size. Then, a neural network-based recognizer takes a temporal representation of the input word and identifies each of its letters (alphabets) without performing an explicit segmentation step. Finally, the predicted word is compared with all possible matches in the reduced lexicon using a customized string-to-string similarity metric.

The structure of the cursive word recognition system is shown in Figure 16. The system is composed of three major modules: Preprocessing, Filtering and Recognition. A preprocessing module (Chapter 4) is necessary because the output of the digitizing tablet is noisy (due to quantization effects and the shaking of the hand) and usually contains too many points. Furthermore, normalization of different writing orientations, writing slant, and writing sizes is also essential in order to reduce writer-dependent variability.

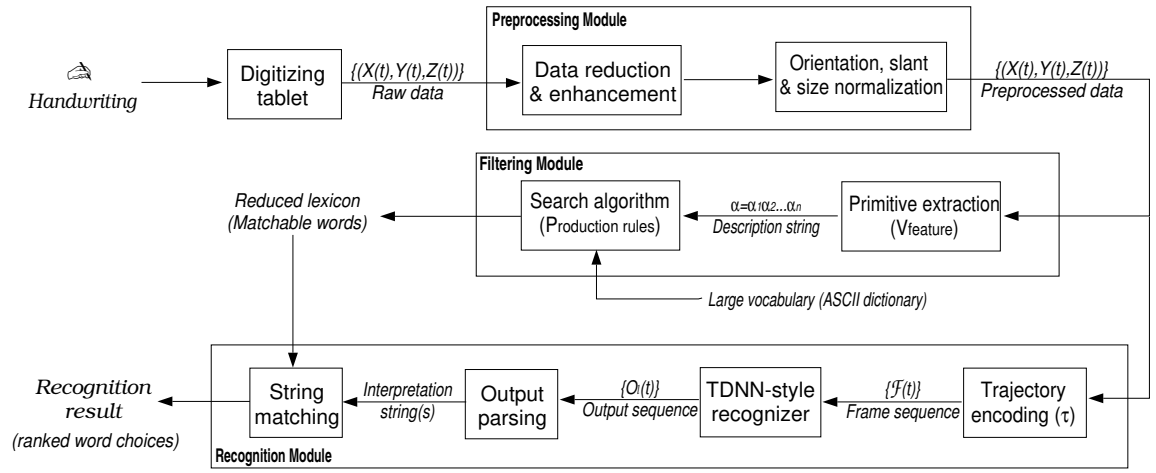


Figure 16: Overview of proposed system for large vocabulary recognition of on-line handwritten cursive words. Three major modules conform the approach: *Preprocessing, Filtering* and *Recognition*.

The Filtering module (Chapter 5) takes a preprocessed word image and extracts a structural description of it in terms of basic features (stroke primitives). The string of (concatenated) stroke primitives, representing the shape of the input word, is then used to derive a set of matchable words. This set consists of words from the system's lexicon that are visually similar to the input word (e.g., the words 'imaginative', 'immigration', and 'imagination' are similar based on coarse shape). The importance of a reduced

lexicon is in limiting the amount of computation required during the string matching – postprocessing – stage (see below). The design of the filtering module was driven by the following goals: (i) robustness with respect to degenerate characters, (ii) flexibility in accommodating variations in writing style, and (iii) computational efficiency (i.e., the need for a very fast procedure of carrying it out). These considerations led us to discard a "template-matching" approach in the derivation process; that is, we do not attempt to match the string of stroke primitives representing an input word against word prototypes. Instead, a set of rules mapping the composition of stroke primitives into English characters is specified. The set of matchable words is then determined by generating all possible letter strings that can be derived from the string of primitives using those rules. The set of matchable words constitutes the reduced lexicon.

The Recognition module (Chapter 6) uses a representation of the input that preserves the sequential nature of the cursive data and justifies the use of a network architecture similar to the Time-Delay Neural Network (TDNN). TDNNs have been successful in learning the temporal structure of events inside a dynamic pattern and the temporal relationships between such events [98]. The neural network-based recognizer is trained to classify the signal within its *fixed-size* input window as this window sequentially *scans* the input word representation, thus bypassing a potentially erroneous segmentation procedure. By training and recognizing characters in "context" (i.e., including a small portion of the word image that precedes and follows the given character) we minimize spurious responses and, to some extent, account for the co-articulation phenomena. Finally, the

recognizer's outputs are collected and converted into an ASCII string that is matched with the reduced lexicon, provided by the Filtering module, using an extended version of the Damerau-Levenshtein metric [11, 58].

In a sense, the Filtering and Recognition modules operate as two independent classifiers based on "semi-orthogonal" sources of information: the first one tentatively recognizes words using a spatial representation of the image, the second one performs letter-analytical tests using a temporal representation instead.

Chapter 4

Preprocessing Module

Preprocessing of the on-line script is a necessary step prior to the recognition process; it is aimed at cleaning the noise present in the input data due to the digitizing device limitations (i.e., noise removal), and reducing writer-dependent variability to a minimum (i.e., normalization). Figure 17 shows an schematic diagram of the Preprocessing module.

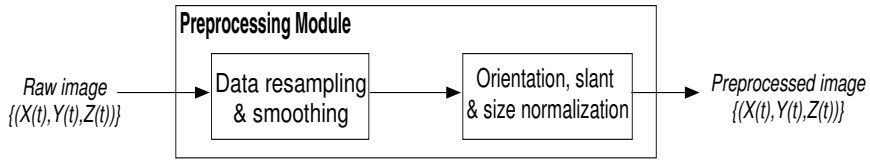


Figure 17: The Preprocessing module: uses a resampling and smoothing algorithm to reduce and enhance the data, and a normalization algorithm to minimize writer-dependent variability.

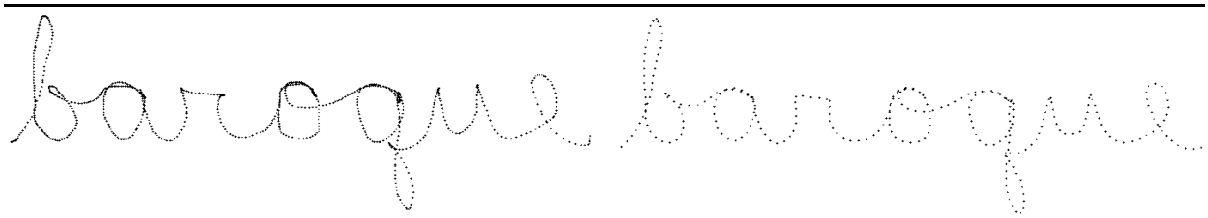
Although numerous preprocessing techniques are reported in the literature [9, 7, 94, 32], the problem remains difficult and far from completely solved. The techniques that are presented here are not new; they were taken from the literature and implemented according to the particular requirements of the available hardware and the specific recognition strategy to follow. Electronic tablets used to record handwritten images operate by periodically sampling (i.e., at a fixed time t) the coordinates of the pen tip movement, $X(t)$ and $Y(t)$. In working with such devices, one is given images with a variable reso-

lution in the space domain; the faster the writer, the fewer the number of points in the on-line representation of the input script. During the recognition process one is generally concerned with capturing the shape of the writing and not with the precise time correspondence of the coordinate points (the opposite may be true in signature verification applications, where the speed of writing is a more difficult characteristic to forge). This being the case, it is generally appropriate to modify the original point sequence in a manner so as to retain only the desired shape information for recognition. Two typical noise removal operations intended for this task are resampling and smoothing. They are also used to reduce noise introduced because of erratic hand motion and inaccuracies of the digitizing device (see Figure 18).

Figure 18: Example of noise present in on-line data due to erratic hand motion and inaccuracies of the digitizing device: (a) an image of character 'A', and (b) a detail of its leftmost vertical stroke (detailed section is indicated with a □).

The resampling operation eliminates duplicated data points (i.e., points recorded at the same location) and reduces (or increases) the number of points by enforcing even spacing between them, resulting in more uniform data. The procedure moves a linear interpolater progressively along the script path, skipping points not sufficiently far from the previous one; when the desired inter-point distance value is exceeded, linear interpo-

lation is used with the skipped points. To avoid "smoothing out" cusps, a test is provided so as to stop the operation before the feature and resume it afterwards. A smoothing operation is then performed by averaging a point with its neighbors — we used the 3-point average $X_{smoothed}(i) = \frac{1}{4}X(i-1) + \frac{1}{2}X(i) + \frac{1}{4}X(i+1)$. In Figure 19 a raw image of the word 'baroque' is shown with the output produced by these preprocessing operations on it.



(a) (b)

Figure 19: Preprocessing example: (a) a raw image of word 'baroque', and (b) the preprocessed image that results after applying the resampling and smoothing routines on it.

The next transformation operations performed on the script are used to normalize the base-line orientation, the slant and the size of words. Base-line correction is intended to translate the orientation of writing to the horizontal level. This operation is important because it affects the efficiency of subsequent processing, such as primitive extraction in the Filtering module. Slant correction (or deskewing) is aimed at removing the oblique or sloping direction sometimes given to characters inside a word. This operation is very desirable because slant is a writer peculiarity which generally does not contain any information for the recognition process. Size normalization is used to further reduce writer-dependent variability by constraining words to be of a specified size.

Our estimation of the base-line location is based on the work of Brocklehurst and Kenward [7]. The algorithm first locates all downward strokes (these correspond to pieces of the drawing between pairs of consecutive local y-maxima and y-minima) in the script and subsequently classifies them according to their vertical extent. The y-extremas corresponding to downward strokes believed to be of median letter height (i.e., the height of lowercase letters without ascenders or descenders) are used to independently find a best fitting straight line approximation to the base-line and half-line. If the orientation of the base-line (base-slope) and half-line (half-slope) differ by less than a threshold (i.e. they are similar), then the the average of the two estimates is used to correct the orientation of the word by rotating it. Otherwise, the estimate based on a larger evidence (i.e. number of y-extremas) is only used and the other one is considered unreliable.

The slant correction algorithm is based on the kinematic approach suggested by Singer and Tishby [88]. The idea is that removing the slant of the script is equivalent to removing the correlation between the horizontal velocity (V_x) and vertical velocity (V_y), and that a measure of such correlation can be easily estimated by $E(V_xV_y)/E(V_yV_y)$ where $E(uv)$ corresponds to the expected value of uv. In our experiments, this approach was significantly faster, and distorted letter shape less than other algorithms based on shear or rotation.

Finally, the size normalization algorithm simply scales a given word image, with respect to the vertical axis, to a specified height $\mathcal{H}$ (currently set at about 3mm) while maintaining the same aspect ratio. The ratio $\mathcal{H}/\mathcal{MLH}$ is used as scale factor, where

$\mathcal{MLH}$ is the median letter height estimate. As a result of this procedure, the height of small letters (those that fall between the base-line and the half-line) is approximately equal across words.

Figure 20 illustrates the output produced by these preprocessing operations when fed with an image of the word 'program'; y-extremas are marked with a box (□) and downward strokes are shown as continuous dark lines.

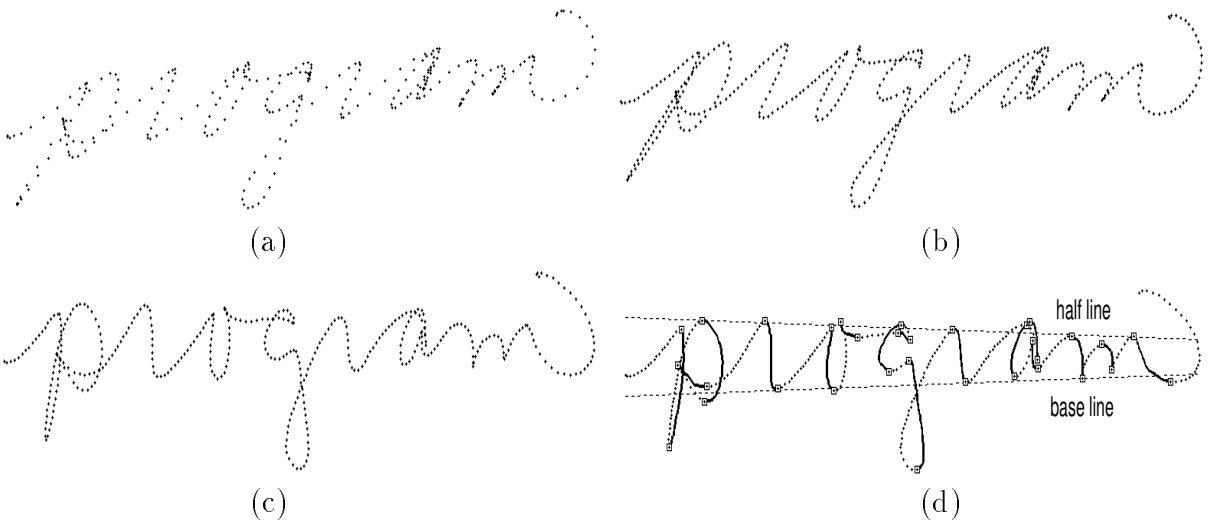


Figure 20: Preprocessing example: (a) a raw image of word 'program', shown after (b) base-line correction and (c) slant correction; in (d) the final preprocessed image is shown with base-line, half-line and extracted downward strokes.

Chapter 5

Filtering Module

In this chapter we describe in more detail the process by which a stroke description string capturing the visual configuration of a word image is computed and how it is subsequently used in filtering/reducing the lexicon (see Figure 21). The vocabulary of the description string corresponds to the different types of downward strokes made by a writer in writing the word. Downward strokes constitute a simple but robust cue that allow for a compact description of the overall shape of a word without having to consider its internal details. Furthermore, they provide formal grounding for the notion of visual similarity, which is the essence of the lexicon-filtering process. Because the operation of the Filtering module can be considered a Syntactic Pattern Recognition approach, we begin with a short overview of this paradigm.

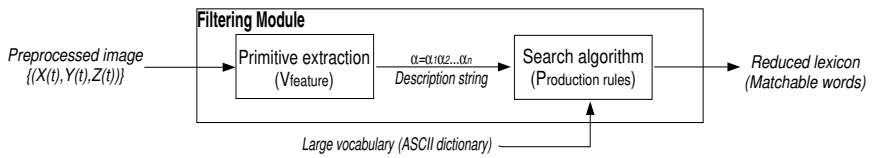


Figure 21: The Filtering module: takes a preprocessed word image and extracts a structural description of it in terms of basic features (stroke primitives) which used for filtering/reducing the lexicon.

5.1 Syntactic Methods in Pattern Recognition

Use of syntactic (structural) methods is one of the major approaches to solving pattern recognition problems [31]. The syntactic approach is applicable to problems where the structure of an object is salient; patterns can be described in terms of simpler subpatterns and each simpler subpattern can in turn be described in terms of even simpler subpatterns, etc. A complex object can then be decomposed into a hierarchy of pattern primitives which can be used for classification and description.

A syntactic pattern recognition system can be looked at through its training and recognition stages. In the training phase, a set of structural elements and their relations is determined from a collection of training images; grammars, or relational models, are generally constructed to represent the structural information exhibited by these elements and their relations. In the recognition phase, the input image is usually preprocessed and then segmented or decomposed to extract structural elements and compute relations among them. A symbolic representation in the form of a string, a tree, or a graph is then derived to describe the structural elements and their relations. Finally, syntax or structural analysis is performed on the symbolic representation to achieve classification and description (see Figure 22). Many successful results have been reported as to applying syntactic methods to a wide range of problems, such as shape analysis, recognition of mathematical equations, chromosome image analysis, texture analysis and character recognition [27].

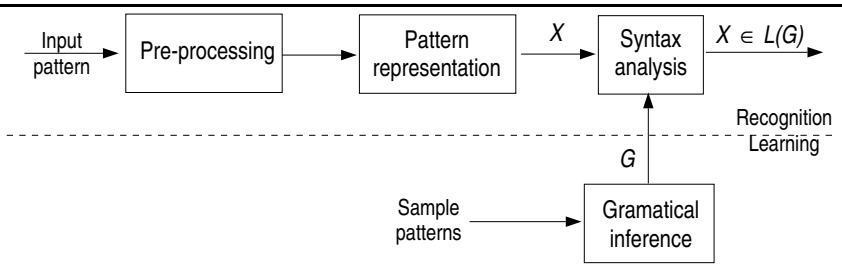


Figure 22: Block diagram of a general syntactic pattern recognition system (from Fu:1977).

Representational schemes and analysis procedures are two major components in the syntactic approach. Representational schemes attempt to give a quantitative representation of the structural information contained in patterns. Grammars and relational models (e.g., graphs) are two formalisms generally used for this purpose. Analysis procedures are used for recognition: the decision on whether or not a given pattern is syntactically correct (i.e., belongs to the class of patterns described by the given grammar or relational structure). Parsing algorithms (or automata) and template matching techniques are commonly employed analysis procedures.

5.1.1 Formal Grammars and Recognition of Languages

Formal grammars have been extensively used to represent pattern classes in the syntactic approach. A grammar G is a four-triple

$$G = (V_N, V_T, P, S)$$

where

V_N is a finite set of *nonterminals*,

V_T is a finite set of *terminals*,

$S \in V_N$ is the start symbol,

and P is a finite set of productions or rewrite rules denoted by

$\alpha \Longrightarrow \beta$ with α and β being strings over $V_N \cup V_T$ (α involving at least one symbol of V_N).

The sets of terminals and nonterminals together correspond to the set of pattern primitives, with the nonterminals being the most basic elements. Production rules in the grammar specify the way of constructing a complex pattern from these pattern primitives. The language generated by grammar G is

$$L(G) = \{x \mid x \in V_T^* \; and \; S \stackrel{*}{\Longrightarrow} x\}$$

That is, the language consists of all strings of terminals generated from the start symbol S. Recognition of languages defined by formal grammars can be carried out by either automata or parsing algorithms.

String grammars are one-dimensional grammars operating on strings of symbols which represent pattern primitives. In this type of grammars, concatenation is the only relation between symbols. Patterns with more complex interconnections need to use higher-dimensional grammars. Examples of higher-dimensional grammars are array grammars, tree grammars, web grammars, plex grammars, shape grammars and graph grammars.

Automata are abstract models of computation devices. An automaton operates on a pattern and accepts or rejects the pattern depending on if the pattern is a member of a

specific language. Automata commonly used are finite automata for regular grammars, and push-down automata for context free grammars.

Parsing is the process of determining if a given string belongs to one of the languages defined by the grammar. If the parser succeeds in parsing, it could provide a sequence of derivations that will indicate how the given string can be derived from the start symbol of the grammar.

Automata or parsing algorithms that are designed on the basis of formal grammars reject patterns that have any errors. In order to deal with imperfect patterns and tolerate some errors, inexact versions of formal grammars along with their parsers and automata have been proposed. Inexact formal grammars incorporate error production rules to allow for the derivation of erroneous patterns. Inexact versions of language recognizers include error-correcting string parsers for string languages and error-correcting tree automata for tree grammars. Another approach to handle distorted and noisy patterns is to use stochastic grammars which incorporate statistical information about the pattern noise and distortion into the recognition process.

5.2 The Task of the Filtering Module

The task of the filtering module is achieved in two steps. The first step is primitive extraction. After this step the input word is represented by a string $\alpha = \alpha_1\alpha_2 \ldots \alpha_n$ of stroke primitives α_i. In the second step, the description string α is passed to a

procedure search(α) which has knowledge about how to derive ASCII letters from the symbols α_i and uses it to generate matchable words. Specifically, a grammar $G_{filter} = (V_{ascii}, V_{feature}, P, S)$ was established, where the set V_{ascii} of terminal symbols is the English alphabet, the set $V_{feature}$ of non-terminal symbols corresponds to the stroke primitives, P is the set of production rules which define the valid combinations of these primitives to generate letters, and S is the starting (or root) symbol. The set of matchable words is then given by the set of strings β which constitute valid English words (based on the original lexicon) and can be derived from α (i.e., $\alpha \stackrel{*}{\Longrightarrow} \beta$).

5.3 Selection of Primitives

According to the harmonic oscillator description of the muscle action involved in handwriting production [66, 40], cursive handwriting generation can be viewed as a sequential modulation of two coupled oscillations; one in the vertical direction and one in the horizontal direction. In this context, it is natural to characterize the writing as an ordered sequence of "upward" and "downward" strokes. However, it has been previously suggested that downward strokes in a word are more important than upward strokes because they are always part of the letters while the former ones sometimes act as joining strokes [7]. Therefore, we choose to use only downward strokes to describe the structure of words.

To identify the primitive strokes, the y-extrema of the preprocessed word image are located (these are local maxima and minima of the y-coordinate in the pen trace) and

the base-line and half-line determined (as described in the chapter on preprocessing). Downward strokes correspond then to pieces of the drawing between pairs of consecutive y-maxima and y-minima; they are extracted and subsequently classified based on: (i) their height and position relative to the reference lines, or (ii) their direction of movement. The current classification scheme identifies 9 different types of strokes; they constitute the elements of $V_{feature}$:

A represents an Ascender stroke (a stroke that extends substantially from the half-line to the upper region of the word);

D represents a Descender stroke (a stroke that extends substantially from the base-line to the lower region of the word);

B represents a stroke that expands Both the upper region and the lower region of the word;

M represents a Median height stroke (a stroke that lies between the half-line and the base-line of the word);

C represents a Connection stroke (a stroke that lies above the center line between the half-line and the base-line);

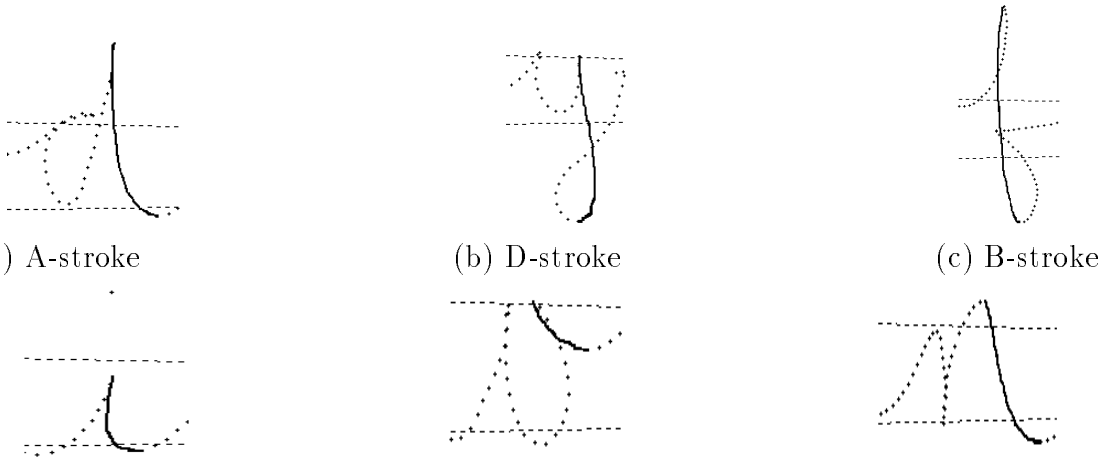
U represents an Unknown stroke (a stroke with an ambiguous classification).

L represents a Left-retrograde stroke (strokes that result from a right-left-right retrograde motion of the pen);

R represents a Right-retrograde stroke (strokes that result from a left-right-left retrograde motion of the pen);

K represents the middle downward stroke in a letter 'k';

Figure 23 illustrates some of these definitions were images of different letters are shown with their corresponding reference lines and relevant downward strokes are indicated as continuous dark lines.



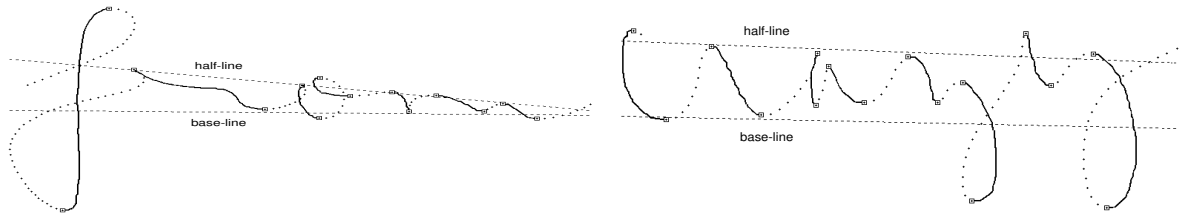
(a) A-stroke (b) D-stroke (c) B-stroke

(d) M-stroke (e) C-stroke (f) U-stroke

Figure 23: Examples of *downward* strokes: (a) an Ascender stroke in a letter 'd', (b) a Descender stroke in a letter 'y', (c) a Both stroke in a letter 'f', (d) a Median stroke in a letter 'i', (e) a Connection stroke in a letter 'o', and (f) an Unknown stroke in a letter 'n'.

Good localization of the reference lines is crucial for the identification of the first five primitive strokes, namely, "A", "D", "B", "M" and "C". This might be considered a limitation since writers are not always consistent in the relative height of letters across a word (e.g., some people tend to write smaller towards the end of the word). To achieve robustness against such variations, the base-line and half-line are not required to be

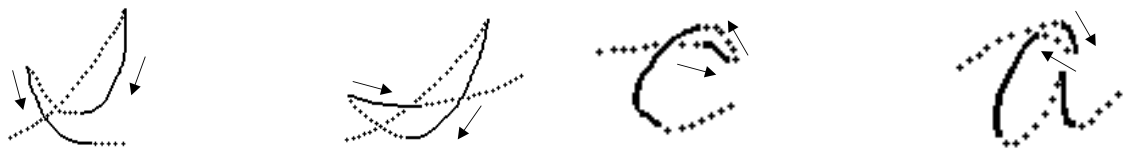
parallel (see Figure 24a). Furthermore, when classifying strokes into these categories, their y-maxima and y-minima are not required to be aligned with the half-line and base-line respectively (see Figure 24b) .



(a) (b)

Figure 24: Examples of downward strokes in word images: (a) a preprocessed image of word 'from' shown with non-parallel base-line and half-line; extracted downward strokes are, from left to right, "B", "M", "M", "C", "M", "M" and "M". In (b), a preprocessed image of word 'crazy' is shown with poorly aligned downward strokes; they are classified, from left to right, as "M", "M", "M", "M", "M", "D", "M" and "D" .

Detection of primitives "K", "L" and "R" is, on the other hand, independent of reference lines location. They are determined by examining the direction of movement in the pen trajectory that precedes and follows the corresponding downward stroke (see Figure 25).



(a) (b)

Figure 25: Examples of *retrograde* pen motion in cursive characters: (a) a (left pointing) retrograde stroke in two different versions of letter 's', and (b) a (right pointing) retrograde stroke in an instance of letters 'c' and 'a'.

Primitive "L" is characteristic of the letters 's' and 'p'; primitive "R" is a peculiarity

of the letters 'a', 'c', 'd', 'g' and 'q'.

Finally, the default symbol "U" is assigned to every stroke which cannot be confidently classified as any of the above categories. Furthermore, since the size of the first character in a given input word is not always consistent with the size of the rest of the word, we relabel as "U" any of the first three downward strokes which was classified as "A", "D", or "B". This is illustrated in Figure 26.

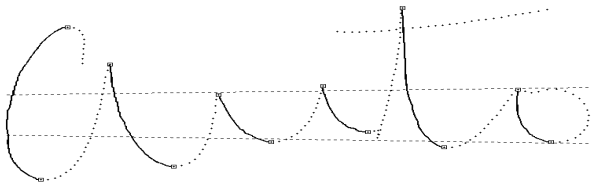


Figure 26: Illustrates the need for the rewrite rule $\alpha_i \rightarrow U; 1 \leq i \leq 3; \alpha_i \in \{A, D, B\}$. A preprocessed image of word 'auto' is shown with base-line, half-line and extracted downward strokes. The size of the first letter is "inconsistent" with the size of the rest of the word.

5.4 Generation of Matchable Words

Having a set of primitives available, the next step is the construction of a grammar G_{filter} that maps the string α of stroke primitives into legal words. Ideally, such a grammar should be automatically inferred from a given set of training samples. Since automatic learning is difficult due to the size of the training corpus required, we resort to intuitive knowledge of cursive character generation. The following questions served as a guideline in the design of the set of production rules P:

1. What stroke primitives are always present in each cursive letter when properly written ?

2. What is the minimum number of stroke primitives that must be detected in a poorly written cursive word to still be able to conjecture the presence of a given letter ?

For example, in a nicely written letter 'w' there should always be three median size ("M") downward strokes. On the other hand, to hypothesize the presence of a letter 'w' in a sloppy written word, at least one median size ("M") downward stroke must be detected. With these ideas in mind, we arrived at a set of 73 production rules; some of these are shown below (a complete listing is presented in Appendix A):

$$
\begin{aligned}
V_{feature} &= \{A, D, M, B, C, K, L, R, U\} \\
V_{ascii} &= \{a, b, c, \ldots, z\} \\
P &= \{A \rightarrow b|d|f|h|k|l|t \\
&\quad D \rightarrow f|g|j|p|q|y|z \\
&\quad M \rightarrow a|c|e|i|m|n|o|r|s|u|v|w|x \\
&\quad U \rightarrow a|b|\ldots|z \\
&\quad \ldots \\
&\quad AM \rightarrow b|h|k \\
&\quad MA \rightarrow d
\end{aligned}
$$

$$RD \rightarrow g|q$$

$$\ldots$$

$$MMM \rightarrow m|w$$

$$RDM \rightarrow q$$

$$S \rightarrow AS|DS|MS|BS|CS|NS|KS|LS|RS|US|\epsilon\}$$

So, for example, the letters 'b', 'h' and 'k' can be described by the primitive's string "AM"; the letters 'm' and 'w' by the string "MMM" and so on. The last production rule is given for completeness but the derivation process is never started from the root symbol (i.e., the grammar is not used as an acceptor).

In general, a description string α does not contain too many "U" symbols. Since such a string can account for a limited number of words in the dictionary, an exhaustive search strategy is adopted to find the corresponding set of matchable words. The search technique uses a trie [49] representation of the dictionary and attempts all possible left-most derivations which transform α into valid English words (see Figure 27). After each step in the derivation, the letter string found up to that point is checked with the trie to determine if it constitutes the prefix of some word. If not, the last step in the derivation is discarded and a different production rule is applied. This process is continued until the end of the symbol string is reached or all possible production rules have been tried in turn.

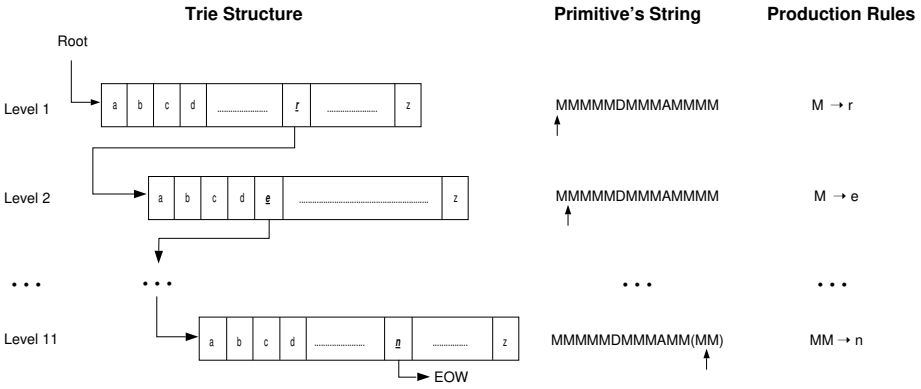


Figure 27: Derivation of matchable words: given a string α of concatenated stroke primitives and the *trie* representation of the lexicon, the search procedure attempts all possible leftmost derivations which transform α into valid English words. In this example the word 'recognition' can be derived from the primitive's string $\alpha =$ "MMMMMDMMMAMMMM".

The final set of matchable words is further pruned if any diacritical mark (points on 'i','j', 't' bars, and 'x' slash) is detected in the input image. Specifically, if a given ASCII candidate word has fewer diacritical marks than what was detected in the input image, the candidate word is discarded.

In Figure 28 an image of the word 'recognition' is shown with its extracted downward strokes and a description of its shape as captured by the string that results of concatenating them. The complete set of matchable (i.e., visually similar) words that can be derived from this string, and a 21k input lexicon, is also shown; there are a total of 17 words in this set. That is, the Filtering module is able to hypothesize that out of 21,000 possible words only 17 match the shape of the given input image. The remaining problem is to make letter analytical tests to determine which of these 17 words is the best match; this is the task of the Recognition module.

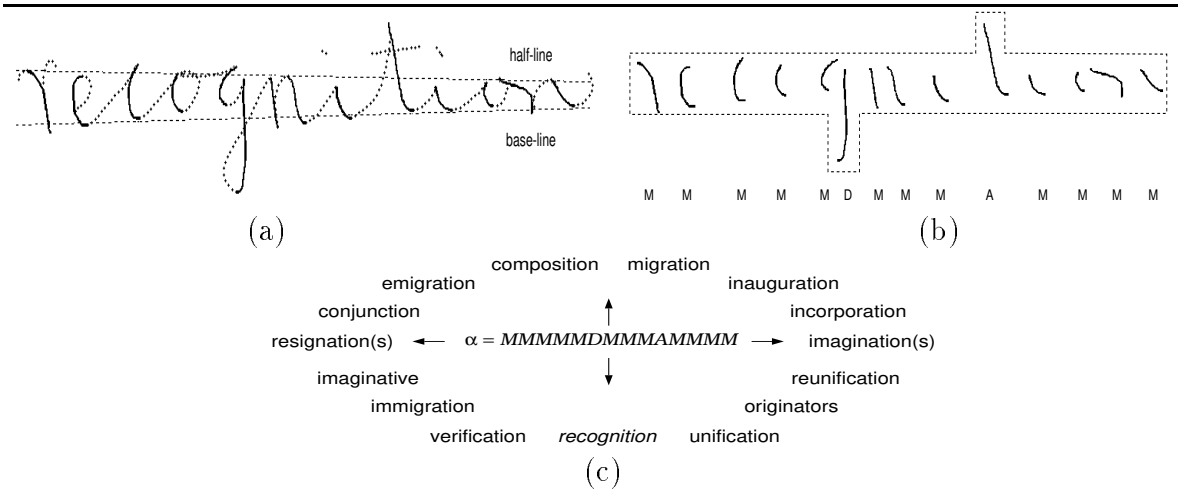


Figure 28: Filtering example: (a) a preprocessed image of word 'recognition' shown with baseline, half-line and extracted downward strokes; (b) the coarse representation of the word-shape provided by the string of concatenated stroke primitives; and (c) the set of matchable words derived from this string ($\alpha = MMMMMDMMMAMMMM$) with a 21k lexicon.

5.5 Testing of Filtering Module

Three different success measures can be used to determine the effectiveness of the filtering module: accuracy, which is the probability with which the correct word appears in the reduced lexicon; reduction efficacy, which measures the relation between the average size (number of words) of the reduced lexicon compared to the original lexicon size; and speed, or the average time taken to carry out the reduction process. A system could thus achieve a 100.0% accuracy by simply making the reduced lexicon equal to the input lexicon; the corresponding reduction efficacy will, however, be 0.0%. Clearly, a successful filtering module must have then both a high accuracy and a high reduction efficacy.

On a database of 3,686 cursive words (1 to 15 letters long) written by 57 different writers, using a lexicon of 21,000 words, the current version of the filtering module outputs

a stroke description string α from which the correct word can be derived in 3092 cases (i.e., 83.88% accuracy). The size of the correctly pruned lexicon was 306 words on average (i.e., 98.5% reduction efficacy) and 6113 words in the worst case. The detailed characteristics of the data used for evaluation of this module are explained in Appendix C.

5.6 Discussion of Filtering Module

It cannot be claimed that the elements of $V_{feature}$ represent an optimal or even a complete set of cursive handwriting primitives. They have been chosen not to allow recognition but rather to obtain a compact but adequate description of the (geometric) shape of the input word image. Furthermore, there is ample evidence for the perceptual relevance of ascending and descending extensions [5]. These primitives are also easy to compute, a necessary condition to meet the speed efficiency requirements.

The performance levels achieved indicate that the selected features offer significant discrimination capabilities. We found, however, no other references for this kind of discrimination so that qualitative comparisons to other techniques are difficult.

Certainly other additional features can be explored. In particular, features, such as convexities and concavities, which are based only on the direction of the trajectory and not on heights are desirable because writers are not always consistent with respect to the relative heights of letters inside a word.

Another potential direction of generalization is to attach weights to the production rules, where higher weights are given to rules with a "stronger" left side. One would then get a ranked reduced lexicon. Furthermore, this weighting constitutes a mechanism for writer adaptability; the system can identify which rules are usually "fire" for a particular writer and increase their weight.

An examination of the images where the Filtering module failed to include the correct word in the reduced lexicon, revealed that most of the errors were due to failures in preprocessing (i.e., in the estimation of the base-line and half-line). This is particularly the case for short words (e.g., 'to', 'of', 'be', etc.) where the number of downward strokes is only 2 or 3, and so the reference lines can not be estimated reliably. A more robust estimator is thus needed for short words.

Chapter 6

Recognition Module

In this chapter we describe the neural network-based recognition technique that bypasses the need for an explicit letter segmentation step by exploiting the temporal representation of the input. A further advantage of such a representation scheme is that stroke absences (from unintentional pen lifts) and accidental intersections (i.e., overlapping or touching characters) which significantly alter the topological (static) pattern of the word, have little or no influence on the dynamic pattern of it. We also present a generalization of the Damerau-Levenshtein string difference metric, which is used to integrate the output of the Recognition module with that of the Filtering module.

The task of the Recognition module is accomplished in four steps (see Figure 29). The first step is the encoding of the pen trajectory as a sequence of frames $\mathcal{F}(t)$ (a frame denotes one discrete time step's worth of data — features). In the second step, a TDNN-style network operates on a window of frames (comprising a character and parts of its neighbors) and produces an output at every time interval. In the third step, a postprocessor interprets this output sequence to generate a letter sequence (interpretation string). Finally, in the fourth step, a string distance algorithm is used to match the

interpretation string(s) with the reduced lexicon produced by the Filtering module.

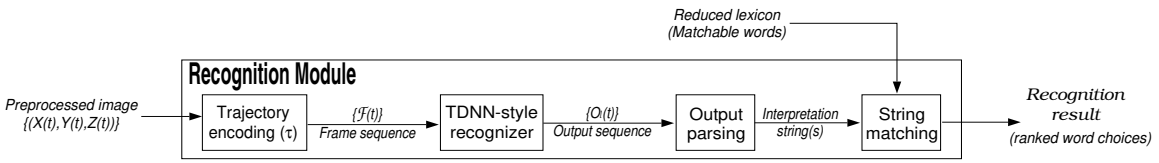


Figure 29: The Recognition module: takes a preprocessed word image and a (reduced) lexicon as input, and produces a ranked list of word choices as output.

We begin with a short overview of the neural network paradigm where we attempt to highlight some key concepts of this technology.

6.1 Artificial Neural Networks

Biologists estimate that the human brain has about 10^{11} neurons (nerve cells) each connected to about $10,000$ other cells [12]. A typical biological neuron has three major regions: the cell *body*, the *axon*, and the *dendrites*. The axon is a long branching fiber that carries signals away from the neuron (i.e., output), and the dendrites consists of more branching fibers that receive signals (i.e., input) from other nerve cells via synapses. Cell bodies can act as information processors: incoming signals raise or lower the electrical potential inside the body of the receiving cell; if this potential reaches a threshold, a pulse or action potential is sent down the axon (it is said that the cell has "fired"). It is believed that the brain's computational power is derived from a massively parallel system where the number of computational units (i.e., neurons) is large, their connectivity is severely restricted (usually to be very local), and their internal complexity is limited.

The high performance of the biological neural system on such complicated tasks as vision and speech understanding provides motivation to consider this computational mechanism for automated pattern recognition applications.

McCulloch and Pitts [65] proposed one of the earliest models of an artificial neuron as a binary thresholding device. Specifically, the neuron computes a weighted sum of its inputs, and outputs a one or a zero depending on whether this sum is above or below a given threshold (see Figure 30):

$$Out_j(t+1) = \Theta(\sum_{i=1}^{n} \omega_{ji}\xi_i(t) - \mu_j)$$

where

$\Theta(x)$ is the unit step function

$$\Theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

ω_{ji} corresponds to the synapse connecting neuron j to input i. The connection is said to be *excitatory* or *inhibitory* depending on whether it is positive or negative. μ_j is the threshold value that must be reached or exceeded for the unit to fire. Real neurons are of course more complicated, but McCulloch and Pitts proved that a synchronous assembly of such neurons is capable of "universal computation" for appropriately chosen set of weights ω_{ji} (i.e., it can perform any computation that an ordinary digital computer can) [37].

Around 1960, Rosenblatt [78] proposed the *perceptron* architecture, composed of layers

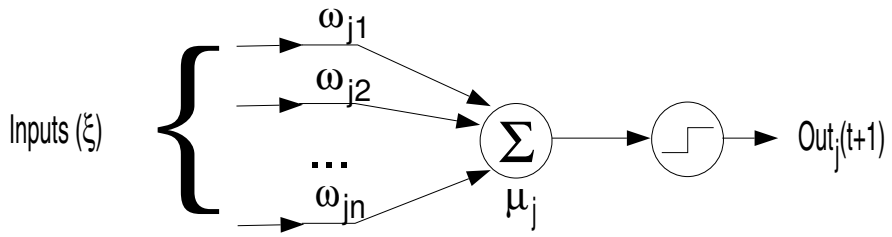


Figure 30: Block diagram of a McCulloch-Pitts neuron. The neuron fires if the weighted sum $Out_j(t+1) = \sum_{i=1}^{n} \omega_{ji}\xi_i(t)$ of the inputs exceeds the threshold μ_j.

of units with feed-forward (unidirectional) connections between one layer and the next. An example is shown in Figure 31. A similar network, the *adaline* architecture — *ada*ptive *li*near *ne*uron — was invented by Widrow and Hoff [100] which, like the perceptron, uses a hard thresholding function.

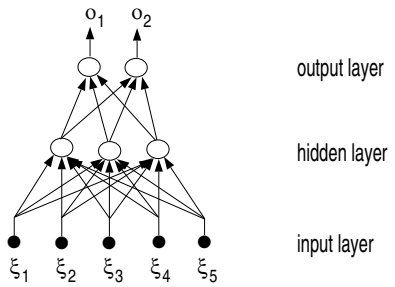


Figure 31: A two-layer perceptron with 5 input units (ξ_i) and two output units (o_i). Only one layer of weights was adjustable in the original perceptron formulation.

For the simplest class of perceptrons (i.e., only one layer of weights is adjustable), Rosenblat was able to prove the convergence of a (supervised) learning algorithm, which corrects the weights iteratively so that the network produces the desired output using a set of training examples. Specifically, given a set of p labeled patterns:

$$\{(\xi^\mu, \tau^\mu); \quad 1 \leq \mu \leq p\}$$

where τ^μ is the desired response to input vector ξ^μ, the problem is that of finding appropriate weights to make the actual output vector o^μ to be equal to τ^μ; it is formulated as a problem of minimizing the *perceptron criterion function* [14]:

$$J(w) = \begin{cases} \sum_\xi (-w^t \xi) & \xi \in \text{set of misclassified patterns} \\ 0 & \text{otherwise} \end{cases}$$

with $\nabla J(w) = \sum_\xi (-\xi)$. The basic gradient descent procedure prescribes then to start with some arbitrarily chosen weight vector w^0 and compute the gradient $\nabla J(w^0)$; the next value, w^1, is obtained by moving some distance from w^0 in the direction of "steepest descent" (i.e., along the negative of the gradient). In general,

$$w^{k+1} = w^k \; - \; \rho \nabla J(w^k) = w^k \; + \; \rho \sum_\xi \xi$$

where ρ is a positive scale factor (or learning rate). If the input patterns are "linearly separable", the sequence of weight vectors will terminate at a solution vector after a finite number of corrections.

The optimism created by this early success was soon dispelled when Minsky and Papert [67] pointed out that some rather simple problems, such as computing the *XOR* function, were not linearly separable and hence could not be solved by the single-layer perceptron. Though it was believed that more layers of units would suffice to overcome this limitation, no learning algorithm was known for such a multi-layer architecture. Minsky and Papert judged the extension to be "sterile". Given his prestige, Minsky's observations were an influential factor for many researchers to leave the field of artificial neural networks for almost 20 years.

In the mid-70's, Werbos [99] presented the conceptual basis of the back-propagation algorithm, a gradient descent technique capable of adjusting the weights in multi-layer perceptrons. But it was not until the mid-80's, when the algorithm was rediscovered by Rumelhart et al. [80], that its use became widespreaded.

6.1.1 The Backpropagation Algorithm

The basis of the Back-Propagation (BP) learning rule is again gradient descent and the chain rule. It requires units with differentiable thresholding functions, with the sigmoid function being a common choice

$$f(x) = \frac{1.0}{1.0 + e^{-\lambda x + bias}}$$

where λ is the gain parameter that can be used to control the "steepness" of the output transition and *bias* is the offset parameter that can be used to adjust the "position" of the function. Transfer functions of this type, with a central high-gain region and decreasing positive and negative gain regions, offer a solution to the *noise-saturation dilemma*: neurons must handle small inputs (which require high gains) as well as large inputs (which should not saturate the output).

The most popular error measure, or cost function, used for optimization this time is the least mean square criterion

$$E(w) = \frac{1}{2} \sum_{\mu i} (\tau_i^\mu - o_i^\mu)^2$$

which is clearly a continuous differentiable function of every weight. We can think of $E(w)$ as a complicated surface above the space spanned by all weights in w; this surface is known as the error surface of the network, and what we are looking for is a global minimum in this surface. The advantage of the mean squared error measurement scheme is that it ensures that large errors receive much more attention than small errors. Furthermore, it is more sensitive to errors made on commonly encountered inputs than it is to errors made on rare inputs [36].

For the hidden-to-output connections the gradient descent rule gives (see Figure 32)

$$\Delta w_{kj} = \lambda \delta_k Out_j \quad where \quad \delta_k = (\tau_k - o_k) f'(S_k), \quad o_k = f(S_k) \quad and \quad S_k = \sum_v \omega_{kv} Out_v$$

and for the input-to-hidden connections,

$$\Delta w_{ji} = \lambda \delta_j \xi_i \quad where \quad \delta_j = (\sum_k \delta_{kj} \omega_{kj}) f'(S_j)$$

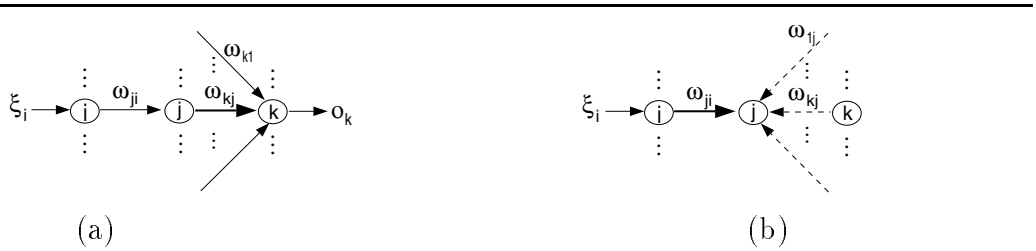


Figure 32: Schematic diagram of the back-propagation weight update rule for (a) a hidden-to-output connection, and (b) input-to-hidden connection.

The above update rules are sometimes written as sums over all patterns μ, and weights are only changed after all patterns in the training set have been presented (batch mode). Learning after each example (ξ^μ, τ^μ), as opposed to learning with respect to the complete training set, is usually superior (i.e., faster) when the training set is highly regular

or redundant. BP suffers from the same drawbacks as many other mean square error procedures: can be exceedingly slow to converge, and can get stuck at a local minima. The method, however, can deal with very large numbers of parameters (weights), larger than can be reasonably handled by more direct methods.

BP networks have proven to be very competitive with classical pattern recognition methods, especially for problems requiring complex decision boundaries [42]. The ability of BP networks to deal directly with large amounts of low level information rather than higher-order (more elaborated) feature vectors has also been demonstrated in different applications (e.g., [46]).

6.1.2 Feed-forward Networks and Pattern Recognition

It is a well established result in Pattern Recognition that all that a pattern classifier needs to know in order to make an optimal classification decision for a given input ξ, in a k-class problem, is the vector of *a posteriori* probabilities

$$p = (prob(\omega_1|\xi) \;\; prob(\omega_2|\xi) \ldots prob(\omega_k|\xi))^T$$

and the scheme of losses with which its decisions are evaluated

$$\lambda_{ij} = cost\ of\ choosing\ class\ \omega_i\ when\ class\ \omega_j\ is\ the\ true\ class$$

Knowledge of λ_{ij} is usually taken for granted (e.g., all errors are equally costly) and thus the problem of building a pattern classifier is that of estimating p from a given learning data set.

However, *a posteriori* probabilities are in turn connected with *a priori* probabilities and *class-conditional* probabilities by means of Bayes rule

$$prob(\omega|\xi) = \frac{prob(\xi|\omega)prob(\omega)}{prob(\xi)}$$

Because the *a priori* probabilities $prob(\omega)$ can be either set to $1/k$ or replaced by plausible estimates, the alternatives for building a classifier are thus to construct approximations for either the

- *a posteriori* probabilities $prob(\omega|\xi)$, or
- *class-conditional* probabilities $prob(\xi|\omega)$

The first approach is ideally suited for functional approximation using a set of *basis* functions. It can be shown that developing regression functions with the objective of estimating τ from ξ (this is the information available to us from the training set) directly results in estimations for $prob(\omega|\xi)$ [83]. The second approach is suited for working with well-known statistical models such as the multivariate normal density functions.

Finally, it is a well established fact that a multilayer feed-forward network with as few as one hidden layer is capable of approximating any continuous multivariate function [41, 92, 30, 83]. Graphically, the first layer (hidden layer) generates the basis functions and the second layer (output layer) implements the linear combination; the weights and the thresholds of the first layer determine the position, orientation and steepness of the basis functions while the weights and the thresholds of the second layer determine the position,

orientation and shape of the resulting "bumps" above ξ-space. By superimposing enough numbers of basis functions, arbitrary landscapes can be formed.

6.1.3 The Time-Delay Neural Network

The Time-Delay Neural Network (TDNN) is a multilayer feed-forward architecture originally devised for the recognition of phonemes "Bee", "Dee", "Ee" and "Vee" using a *spectrogram* (distinguishing between these sounds is considered particularly difficult in speech recognition). A spectrogram is a two-dimensional pattern where the vertical dimension corresponds to frequency and the horizontal dimension corresponds to time (i.e., frames). Figure 33 illustrates a single hidden-layer version of the TDNN [98, 53]; the input units represent a single time frame $\mathcal{F}(t)$ of the spectrogram and the whole spectrogram is processed by scanning it, one frame at a time, with the input units. Each hidden unit has a receptive field that is limited by a time delay (e.g., a unit's decision at time t in the first hidden layer is based on frames $\mathcal{F}(t), \mathcal{F}(t-1), \mathcal{F}(t-2)$); that is, hidden units are connected to a limited temporal window within which they can detect temporal features. Since hidden units apply the same set of synaptic weights at different times, they produce similar responses to similar input patterns that are shifted in time. The construction is further motivated by the observation that the sequence of layers can generate features with an increasing view over the input and hence exhibit increased discriminative power.

TDNNs are trained with a modified back-propagation (BP) algorithm [80] and are

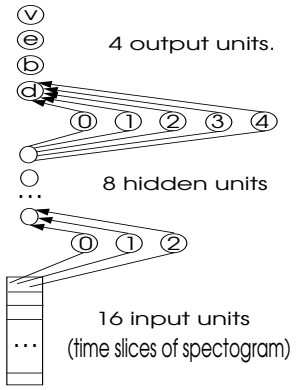


Figure 33: A three-layer time-delay neural network (TDNN) used to recognize phonemes. Hidden units have a receptive field that is limited by a time delay.

usually less difficult to train than (although sometimes outperformed by) recurrent networks [4] for time signal processing.

In designing a neural network based solution to our specific character recognition problem, we decided to employ the TDNN architecture because of its demonstrated ability to learn the temporal structure of events inside a dynamic pattern, training algorithms were available, and it appeared possible to adapt its structure to suit our problem in such a way so that the behavior of units, or groups of units, remained meaningful. The idea that it is possible for the structure of a problem to be reflected directly in the structure of the network has been referred to as the *isomorphism hypothesis* [90] and is depicted in Figure 34.

Each of the main processing steps of the Recognition module — namely, encoding of the pen trajectory, TDNN-style network architecture, interpretation of network's output, and string distance algorithm — is now described.

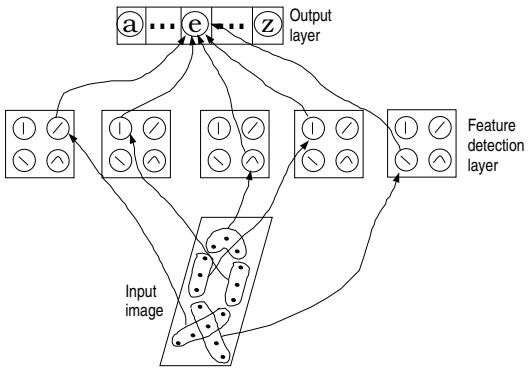


Figure 34: Schematic diagram of a hypothesized feed-forward network for letter identification. A possible set of "feature" detectors (circles) and active ones after presentation of an image of letter 'e'.

6.2 Trajectory Representation

On-line data represents text as a sequence of points $\{P(t) = (X(t), Y(t), Z(t))\}$, where X, Y are the coordinates of the pen tip, and Z indicates pen-up/pen-down information. All relevant dynamic information about handwriting can presumably be inferred from this sequence but this data is too unconstrained; more efficient methods of encoding it must be employed. At the same time, we want to avoid subjectivity in selecting features, a process which could result in the discarding of information essential for recognition. Therefore, we choose mainly to encode information pertaining to local direction and curvature in the pen trajectory, and rely on the neural network-based recognizer for the selection of features relevant to performing the classification task.

Chain coding [24] is a technique frequently used to encode direction in a connected sequence of points. However, one problem with this one dimensional representation is

that false discontinuities arise in the coded-direction domain. We avoid this problem by using two parameters in our trajectory representation: (i) $\sin\theta_y(t)$ - sine of the angle between each segment $\overline{P(t-1)P(t+1)}$ of the trajectory and the Y-axis, and (ii) $\sin\theta_x(t)$ - sine of the angle between $\overline{P(t-1)P(t+1)}$ and the X-axis (see Figure 35). By restricting $\theta_y(t)$ and $\theta_x(t)$ to vary between $-\pi/2$ and $+\pi/2$ we make the parameters unambiguous; a negative value of $\sin\theta_y(t)$ indicates that point $P(t+1)$ is before point $P(t-1)$ (i.e., a backward pen movement was made in going from $P(t-1)$ to $P(t+1)$), and a positive value indicates that point $P(t+1)$ is after point $P(t-1)$ (i.e., a forward pen movement was made). Similarly, the sign of $\sin\theta_x(t)$ indicates whether point $P(t+1)$ is above or below point $P(t-1)$ (i.e., if an upward or downward pen movement was made). A similar representation was used in [33] but the parameters were interpreted differently.

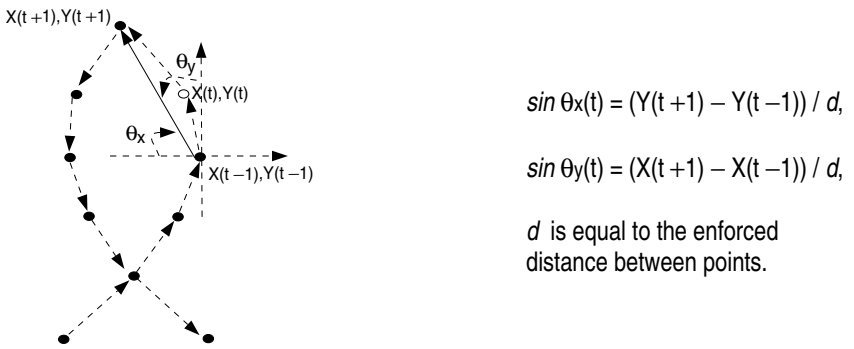


Figure 35: Directional information: An on-line version of a letter 'e', and the parameters used in the encoding of direction in its trajectory.

Although the values of $\theta_y(t)$ and $\theta_x(t)$ could have been used directly, the *sine* function makes them easier to compute, conveniently bounds them between -1 and +1, and provides us with some quantization effect. For instance, small differences in the directional

angles when the pen is describing a jagged "vertical" line going up or down (i.e., $\theta_x(t)$ close to $+\pi/2$ or $-\pi/2$) result in similar values for the upward-downward descriptor. Similarly, small deviations from a straight horizontal line during a forward-backward movement of the pen (e.g., a connecting stroke) result in similar values for the forward-backward parameter. We enhance this behavior by forcing small oscillations about zero of the forward-backward descriptor to be exactly zero. Figure 36 shows the form of the directional parameters for the letter 'w'.

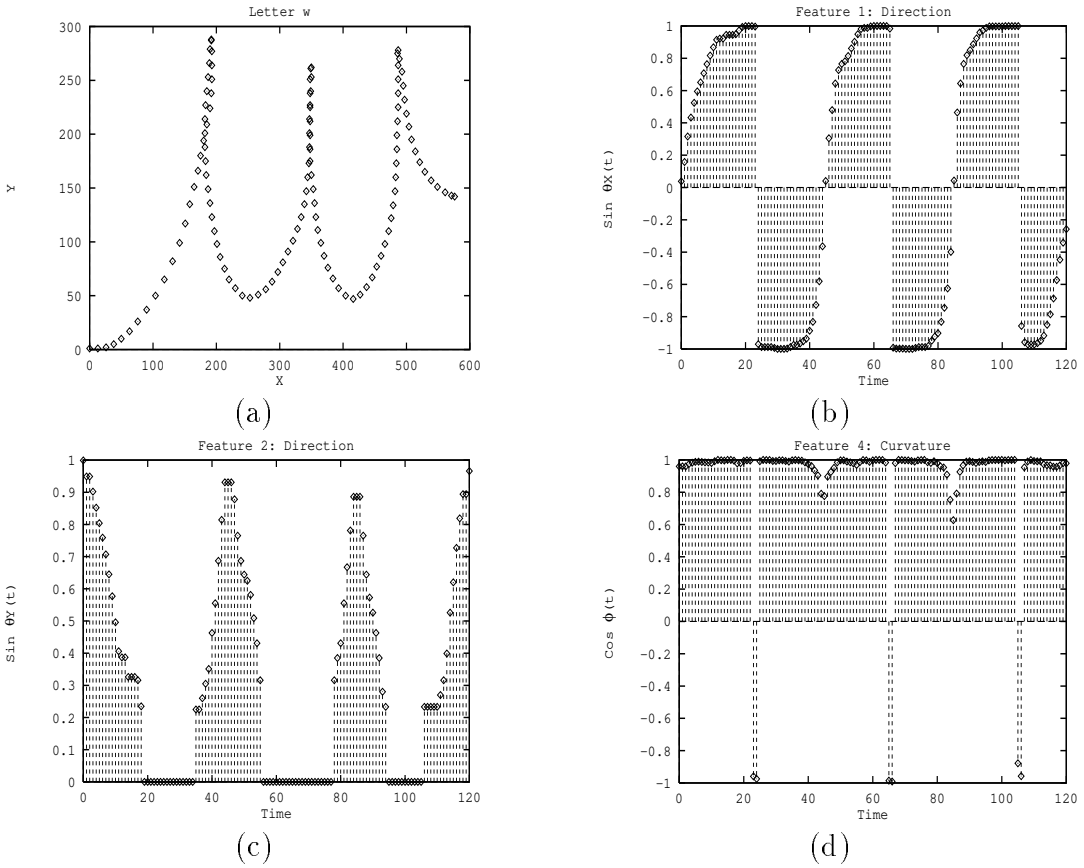


Figure 36: Example of an on-line handwritten letter 'w' shown with the parameters used in the encoding of its trajectory: (a) a letter 'w', (b) the plot of the upward-downward descriptor (i.e.,$\sin\theta_x(t)$), (c) the graph of its associated forward-backward descriptor (i.e., $\sin\theta_y(t)$), and (d) the curvature descriptor (the location of cusps is clearly visible).

In addition to directional information, we also find the location of the points in the trajectory at which sharp changes in the direction of movement (i.e., cusps) take place. A very simple measurement of local curvature can be obtained by calculating the change between two consecutive directional angles. Guyon et al. [33] suggest that the angle $\phi(t) = \theta_x(t+1) - \theta_x(t-1)$ be represented by its sine and cosine values. However, we found that the values of $\cos\phi(t)$ behave more smoothly than those of $\sin\phi(t)$; for small values of $\phi(t)$ (i.e., little change in direction) $\cos\phi(t)$ remains flat at the high value of $+1$ whereas $\sin\phi(t)$ oscillates around zero. We chose $\cos\phi(t)$ as our only curvature descriptor: it goes down to -1 for sharp cusps (independent of their orientation) and down to around 0 for more smoother turns. Figure 36(d) shows the shape of $\cos\phi(t)$ for the letter 'w'; the presence of three cusps is clearly noticeable.

6.2.1 Zone Encoding

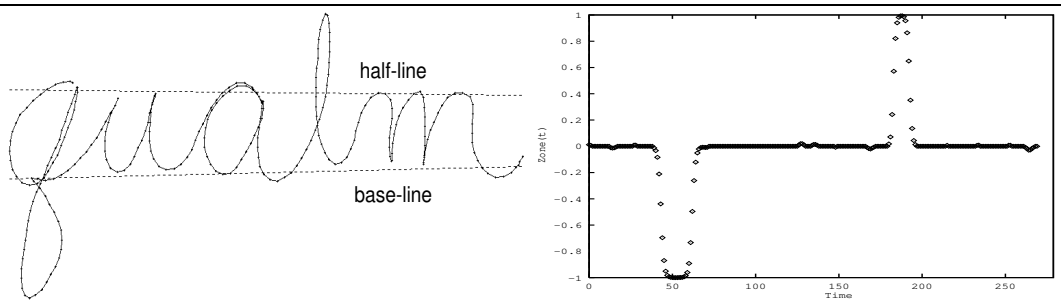
An additional parameter, $zone(t)$, is introduced in the encoding of the pen trajectory to help distinguish between letter pairs such as 'e'-'l', which have similar temporal representation in terms of direction and curvature alone. These pairs can be more easily differentiated by encoding their corresponding $Y(t)$ coordinate values into the previously determined zones: the *middle zone* (between the base-line and the half-line), the *ascender zone* (above the half-line) and the *descender zone* (below the base-line).

For a point $P(t)=(X(t),Y(t))$ falling within the middle zone, we make $zone(t) = 0$; otherwise, we have $0 < zone(t) \leq 1.0$ if the point falls within the ascender zone, and

$-1.0 \leq zone(t) < 0$ if the point falls within the descender zone; specifically, the $zone(t)$ parameter is computed by passing the value of the vertical distance ($dist$) between point $P(t)$ and the half-line (or base-line) through a thresholding function:

$$zone(t) = f(\frac{10.0dist}{body_hght} - 5.0)$$

where $f(x)$ is the sigmoid function; $body_hght$ corresponds to the distance between the base-line and half-line so that when point $P(t)$ is further away than $body_hght$ from the half-line (or base-line), $zone(t)$ is 1.0 (or -1.0). In Figure 37, an image of the word 'qualm' is shown with its base-line and half-line, and the corresponding $zone(t)$ parameter. This coding scheme appears robust against writing distortions where ascenders/descenders are made atypically large or when medium-size letters do not fully fall within the reference lines.



(a) (b)

Figure 37: Zone encoding of the pen trajectory: (a) a preprocessed image of word 'qualm' shown with estimated base-line and half-line, and (b) the associated $zone(t)$ parameter.

6.2.2 Time Frames

Given a sequence $\{(X(t), Y(t), Z(t))\}$ of on-line data, we define a time frame $\mathcal{F}(t)$ to be a 4-dimensional feature vector consisting of four elements:

$$\mathcal{F}(t) = (\sin\theta_x(t), \sin\theta_y(t), \cos\phi(t), zone(t))$$

where the first two elements encode direction, the third element encodes local curvature, and the fourth element encodes zone information.

The frame sequence $\{\mathcal{F}(t)\}$ constitutes an intermediate representation of the on-line data and is used as the input to our neural network recognizer.

6.2.3 Varying Duration and Scaling

Since we are dealing with unsegmented words, a constant number of frames per letter across a word or across a set of samples cannot be guarantee (i.e., varying duration). To reduce such variability in letter length, the size normalization step of the preprocessing module uses the ratio $\mathcal{H}/\mathcal{MLH}$ as scale factor; $\mathcal{MLH}$ – median letter height – is an estimate of the height of small letters (i.e., those that fall between the base-line and the half-line), and $\mathcal{H}$ is the normalization height (currently set at about 3mm). Because the distance between points is kept constant, the above procedure effectively minimizes time distortions of letters.

6.3 Neural Network Recognizer

Multiple decisions have to be made a priori in the design of a TDNN-style network, including the number of layers, size of input, and choice of *delay* connections. The architecture of our three layer[1] TDNN-style network is inspired by that of Waibel *et al.* [98] for phoneme recognition and that of Guyon *et al.* [33] for uppercase handprinted letter recognition. The overall structure of one of the best networks we found is shown in Figure 38.

The choice of $L = 96$ frames as the length of the input window to the network (network receptive field) is related to $\mathcal{H}$, the normalization height. $\mathcal{H}$ is selected as small as possible so as to minimize the convolution time needed to do full word recognition. Having $\mathcal{H}$ available, L is selected so that L frames are enough to represent a character and, in most cases, include part of the characters on each side of it for contextual information. The length of the two hidden layers is then determined using an *undersampling* factor of 3, a technique that allows to reduce the size of the network [55]. This leads to the notion of a *pyramidal* structure in which the input image is recognized at varying levels of detail [3]. To compensate for the loss of resolution associated with undersampling, a commonly used approach is to increase the number of hidden units as one moves up the network pyramid.

The weight connection in the network is arranged such that each hidden unit has

[1] See Lapedes and Farber [54] for a proof that two hidden layers are enough to encode arbitrary decision surfaces.

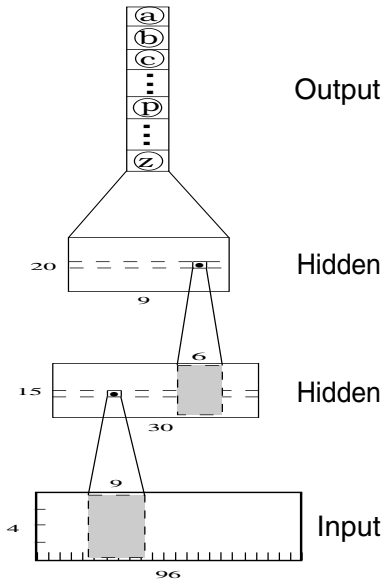


Figure 38: The architecture of a TDNN-style network for cursive word recognition. The net has two hidden layers, an input layer consisting of 96 time frames, an output layer of 26 units, and 7081 independent weights. The first hidden layer consist of 15×30 units, each of which is connected to a window of 9 time steps. The second hidden layer consist of 20×9 units, each of which is connected to a window of 6 time steps.

a receptive field that is limited along the time domain. In the first hidden layer there are 15 units replicated 30 times (i.e., weights are *shared*), each receiving input from 9 consecutive frames in the input layer. The choice of 9 as the width of the receptive field of these units reflects the goal of detecting features with short duration at this level, but also long enough for each unit to detect a meaningful feature (e.g., a cusp). The receptive fields of two consecutive units in the first hidden layer overlap by 6 frames. In the second hidden layer, there are 20 units replicated 9 times, each looking at a 15×6 window of activity levels in the first hidden layer. These units receive information spanning a larger time interval from the input, and hence are expected to detect more complex and global features (i.e., longer in duration). The receptive fields of two consecutive units in the

second hidden layer overlap by 3 frames. Finally, the output layer has 26 units (one for each of the English letters) fully connected to the second hidden layer.

Weight-sharing is a general paradigm that allows us to build reduced size networks [55]. It is commonly believed that minimizing the number of free parameters in the network (i.e., weights that must be determined by the learning algorithm) is an effective way of increasing the likelihood of correct generalization. Furthermore, such weight reduction has been successfully employed for different complex classification tasks without reducing the computational power of the network [46, 47, 63]. Weight sharing also enables the development of shift-invariant feature detectors [80] by constraining units to learn the same pattern of weights as their neighboring ones do. This corresponds to the intuition that if a particular feature detector is useful on one part of the sequence, it is likely to be useful on other parts of the sequence as well. This is true particularly if such feature appears in the input displaced from its ideal, or expected, position.

6.4 Neural Network Simulation

We choose the activation range of our neurons to be between -1 and $+1$ with the following computationally efficient activation function [18] :

$$f(u) = \frac{u}{1+|u|} \text{ with derivative } f'(u) = \frac{1}{(1+|u|)^2} + offset$$

where $|u|$ stands for the absolute value of the weighted sum and $offset$ is a constant suggested by Fahlman [19] to kill flat spots. Weights are initialized with random numbers

uniformly distributed between -0.1 and $+0.1$. A single bias unit is used by all weight-shared units that are controlled by the same weight kernel, as opposed to an independent bias per unit (we found no reason to have independent bias units in order to develop truly invariant feature detectors).

The use of *error tolerance*[2] during training was found to be very helpful in mediating the disproportion between training samples with negative target values (negative evidence indicating that the network should not respond) and training samples with positive target values (positive evidence indicating the network should respond). We started this parameter at 0.3 and subsequently gradually reduced it to 0.1. All simulations were performed with a simulator written in ANSI C.

6.4.1 Training Signal

Each word sample in the training data set was labeled with the positions of each inter-character boundary (roughly where one character ends and the next one begins). This information was then used to pair each frame $\mathcal{F}(t)$, in the dynamic representation of the word, with an output vector. The goal was to generate a target signal that ramps up about halfway through the character and then quickly backs down afterwards (see Figure 39), in such a way that the network learns to recognize a character whenever the center of the character is in the center of the network's receptive field.

[2]An error tolerance of, say, 0.3 means that any activation value of an output unit below -0.7 is considered to be a -1.0 and any value above $+0.7$ is considered to be a $+1.0$ (i.e., no error is fed back).

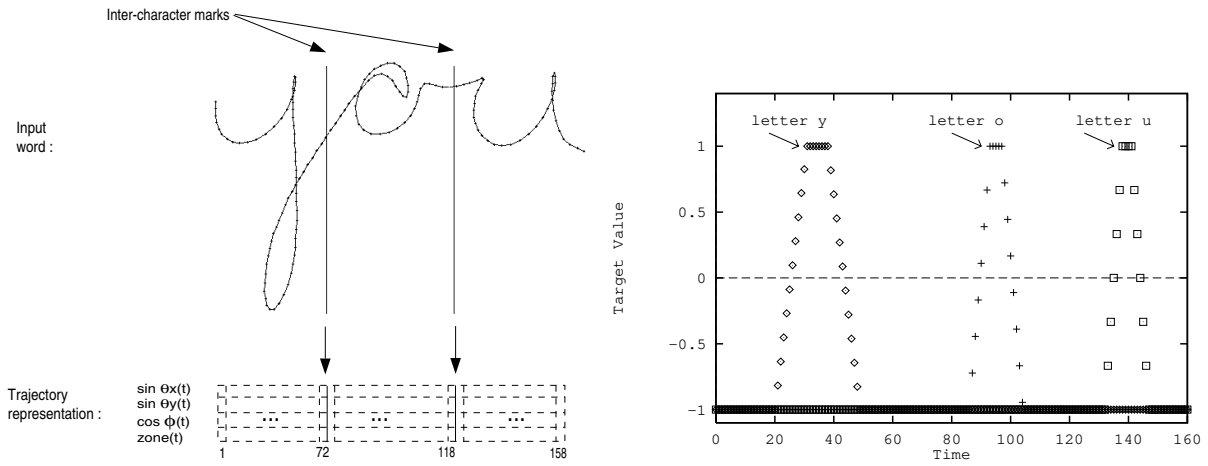


Figure 39: The procedure for generating target vectors for training patterns: (a) the word 'you' is displayed after preprocessing and with intercharacter boundaries marked. The location of these marks is indicated in the dynamic representation of the word. The generated target signals for the letters 'y','o', and 'u' are presented in (b). All other letters have a target value of -1.0 for all frames $\mathcal{F}(t)$.

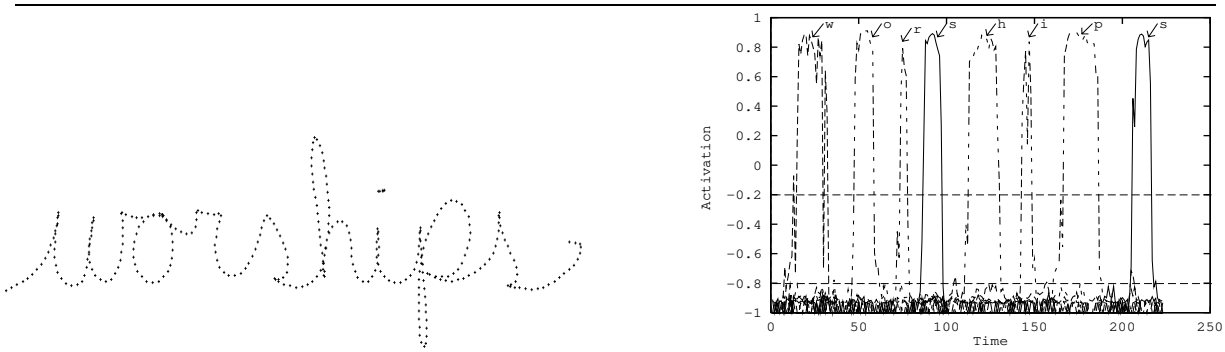
For each word in the training data set, a target signal that ramps up at 30% of each character's length, reaches its maximum between 45% and 55% of the character length, and subsequently backs down to its minimum was generated.

6.5 Output Trace Parsing

Full word recognition is achieved by continuously moving the input window of the network across the frame sequence $\{\mathcal{F}(t)\}$ representation of a word, thus generating activation traces $\mathcal{O}_l(t)$ at the output of the network, where $\mathcal{O}_l(t)$ corresponds to the network's confidence in recognizing a letter l at time t. These output traces are subsequently examined to determine the ASCII string(s) best representing the word image. The input

window is shifted by $\mathcal{S}=3$ frames between successive generations of the output activation trace.

Output trace signals $\mathcal{O}_l(t)$ are inspected looking for activation peaks. Activation peaks are determined by scanning the traces, from left to right, looking for activation values that exceed a given detection threshold (D_THRESHOLD). When the activation value of a letter exceeds this threshold (currently set at -0.8), a summing process begins for that letter that ends when its activation value falls below the threshold. Activation peaks with a maximum value below N_THRESHOLD (currently set at -0.2) are not considered sufficiently "strong" and therefore discarded. The resulting set of activation peaks $\{P_i\}$ is then ordered based on the beginning time of each peak P_i. Figure 40 shows the output activation traces $\mathcal{O}_l(t)$, for all the 26 output nodes, generated by the network when presented with the word 'worships' from our training data set. Eight different activation peaks are clearly visible, each one corresponding to a letter in the word.



(a) (b)

Figure 40: Output activation traces generated by the neural network recognizer: (a) the preprocessed image of a word 'worships', and (b) the plot of all 26 output node responses (i.e., $\mathcal{O}_l(t), l \in \{a, b, \ldots, z\}$) when the network is presented with this word.

Each activation peak is characterized by the following parameters:

begin-time when the corresponding output trace signal exceeds D_THRESHOLD;

end-time when the corresponding output trace signal comes below D_THRESHOLD again;

size area under the peak;

net-size area minus area shared with overlapping peaks;

normalized-size area normalized by its expected value, which given by the average size of all the peaks in the training signal for that letter;

width defined as $MAX(aw, epw)$, where $aw = end-time\ -\ begin-time$ is the actual peak's width and epw is the expected peak width.

The normalization of the size value is required in order to compensate for smaller letters, which are shorter in the temporal domain [35]. The definition of peak width is motivated by the process of determining whether two peaks, adjacent in the time ordering, overlap.

Two additional parameters control the pruning of "false positive" responses of the network: LOW_CONFIDENCE_PEAK and NARROW_PEAK. The former is the minimum value for a peak's normalized-size; the latter is the minimum value for the ratio aw/epw for a peak not to be discarded.

When no two adjacent activation peaks (adjacent refers to the above ordering) overlap each other (as is the case in Figure 40), the output ASCII string is obtained by simply

concatenating the letters represented by each peak. In the more general case, peaks can overlap, requiring a more complex scheme than concatenation. A directed interpretation graph is constructed from the ordered set of activation peaks as follows: there is a node N_i in the graph for every activation peak P_i, and there is an edge between nodes N_i and N_j $(i < j)$ if peaks P_i and P_j are adjacent and their widths do not overlap; otherwise, nodes N_i and N_j will lie on parallel paths of the graph. Figure 41(b) shows the output activation traces for all the output nodes generated by the network when presented with an image of the word 'vainer'. All activation peaks that reach a maximum value above N_THRESHOLD are shown with their expected widths ($\leftrightarrow$) centered around the middle of each peak. Figure 41(c) shows the associated interpretation graph: the number next to each node is the normalized-size of the corresponding activation peak. Word hypotheses are generated by traversing all possible paths in the graph from the root to all the "leaves". The confidence of a word hypothesis is set using the average of the node's normalized sizes in the corresponding path.

6.5.1 Missing Peaks

Sometimes it is possible for the peak parsing routine to "hint" that a character is missing in the output interpretation string. A missing character in the output interpretation string is usually the result of a poorly written character in the input image which results in a low-activation peak which is considered noise or simply discarded because of low confidence during the peak identification process. A frequent consequence of this situ-

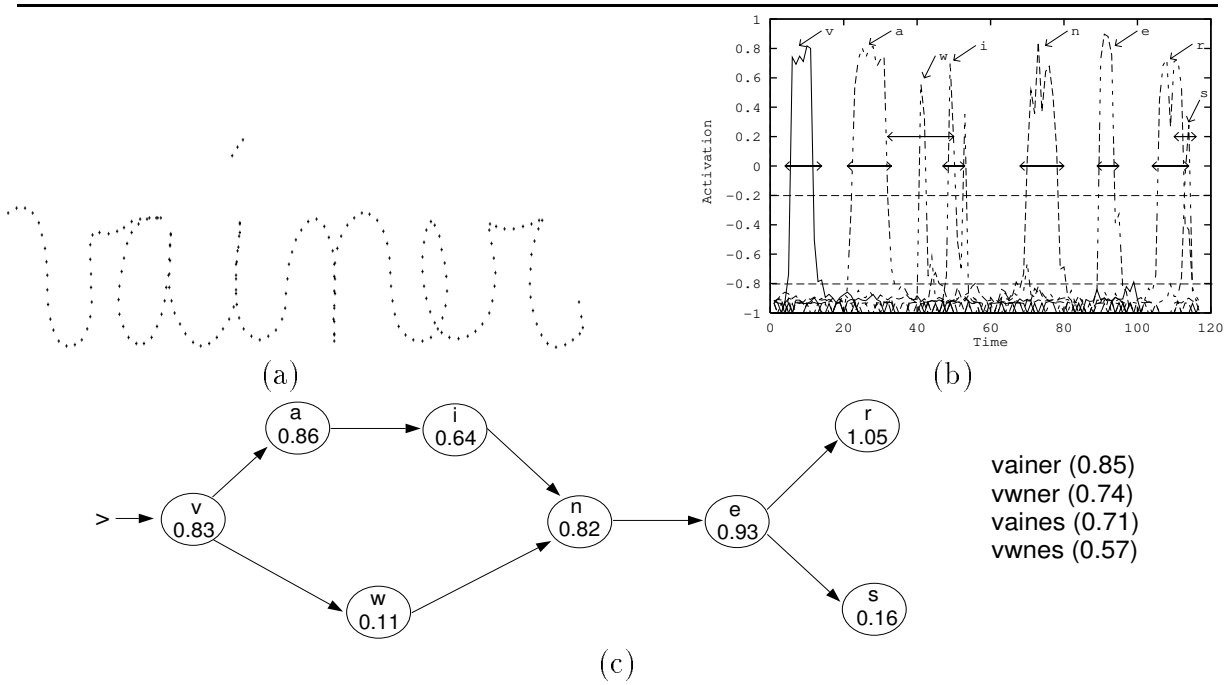
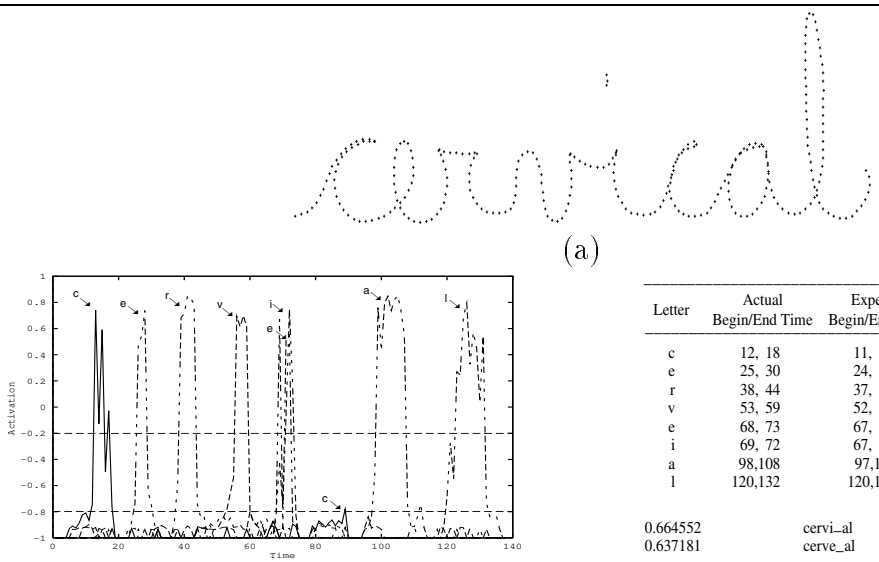


Figure 41: The operation of the output trace parsing algorithm: (a) a preprocessed image of the word 'vainer', (b) the plot of the corresponding network output traces (selected activation peaks are shown with their expected peak width), and (c) the associated interpretation graph and generated word hypotheses (nodes of the graph are shown with their corresponding peak's sizes).

ation is that there will be an unusually large "no-response" time interval in the output activation traces; that is, a period of time for which no $\mathcal{O}_l(t)$ is active.

To detect these cases we have computed the expected inter-peak gap, from our training data set, for every pair of characters. Then, during the traversal of the interpretation graph, if the time-gap between two adjacent activation peaks is larger than its expected value, a special symbol ('_') is output to indicate that a character is probably missing. When matching an interpretation string containing symbol '_' with a lexicon entry, any character is allowed to match '_' with a small penalty. Figure 42(b) shows the output

activation traces generated by the network when presented with an image of word 'cervical'; the missing activation peak for a letter 'c' is noticeable. Figure 42(c) shows the full output produced by the peak parsing routine.



(a)

(b)

Letter	Actual Begin/End Time	Expected Begin/End Time	Size	NetSize	Normalized Size
c	12, 18	11, 19	4.78	4.78	0.44
e	25, 30	24, 30	4.43	4.43	0.56
r	38, 44	37, 45	8.14	8.14	0.81
v	53, 59	52, 60	6.24	6.24	0.53
e	68, 73	67, 73	3.69	1.22	0.47
i	69, 72	67, 73	4.16	1.69	0.66
a	98,108	97,109	14.06	14.01	0.81
l	120,132	120,132	12.09	12.09	0.85

0.664552 cervi_al
0.637181 cerve_al

(c)

Figure 42: Detection of missing activation peaks: (a) a preprocessed image of the word 'cervical', (b) activation traces, and (c) the corresponding output of the peak parsing routine. Missing peaks are indicated by a special character ('_') in the output interpretation strings.

6.5.2 Delayed Strokes

Diacritical marks such as dots on letters 'i' and 'j', and horizontal bars on letter 't' (and sometimes 'x' slash also) are often written after the whole word was written. These *delayed* strokes constitute an exception to our "dynamic" representation scheme of cursive handwriting because they violate the (strict) time-order of the letter patterns.

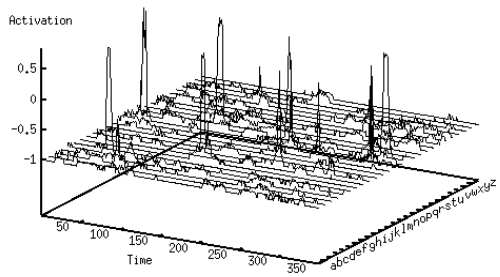
Morasso *et al.* [69] proposed to deal with the problem of delayed strokes using a

re-ordering procedure where they are detected, removed and subsequently inserted next to the point "which is closest in the iconic sense". One difficulty with this approach is that very often it is not obvious to what point of the word the delayed stroke should be linked to; particularly, because these marks are usually carelessly positioned, the closest point in the word may not correspond to the intended location.

Because diacritical marks are many times missing or badly positioned in an image, we decided that they should be used as "confidence boosters" and not as required features for letter identification. That is, the recognizer should be able to hypothesize the presence of a letter 'i', 'j' or 't' in the input script even if the diacritical mark is missing. The existence of a diacritical mark is then simply used to confirm the hypothesis or resolve any ambiguity (say between 'i' and 'e' or between 't' and 'l').

Diacritical marks are thus detected and removed from the image prior to recognition. A (time) region of influence is associated with every detected diacritical mark corresponding to the points in the trajectory "covered" (in a horizontal sense) by the mark. The peak parsing postprocessor was extended to incorporate this information; specifically, a peak for the letter 'i', 'j', 't' or 'x' in the output activation traces is said to be *influenced* by a diacritical mark if some of the corresponding frames in the input trajectory overlap with the mark's region of influence. Confidence of influenced peaks is then boosted by an amount proportional to its current value. In Figure 43 influenced peaks are indicated with a '$*$'.

(a)



(b)

Letter	Input Begin/End time	Output Begin/End time
i	277, 279	206, 256
t	295, 326	253, 275
i	334, 336	263, 313

(c)

Letter	Actual Begin/End time	Expected Begin/End time	Size	NetSize	Normalized Size	
n	21, 30	19, 31	7.73	7.73	0.51	
e	50, 57	49, 57	8.16	8.16	0.93	
o	107,117	106,118	11.88	11.85	0.71	
g	149,165	146,168	9.32	9.32	0.30	
n	204,212	202,214	7.24	7.24	0.48	
m	218,221	210,228	1.55	1.55	0.07	
i	233,239	233,239	3.30	3.30	0.79	*
l	259,264	255,267	1.41	1.01	0.08	
t	257,266	255,267	2.50	2.10	0.17	*
m	319,323	312,330	2.16	2.16	0.10	
n	326,337	325,337	12.78	12.59	0.84	

0.590697 ne_ognit_n

(d)

Figure 43: Example of delayed-stroke processing: (a) a preprocessed image of the word 'recognition' shown with regions of influence corresponding to two i-dots and one t-crossing; (b) plot of network output traces when presented with this image; (c) formal specification of regions of influence for detected diacritical marks, and (d) corresponding detected peaks and generated interpretation string ('ne_ognit_n'); influenced peaks are indicated with a '$*$'.

6.6 String Matching

In order to validate the output interpretation strings produced by the recognizer, we need to look them up in the reduced lexicon that is provided by the Filtering module. Since the interpretation string(s) s often contains errors, a similarity metric is needed to determine the likelihood that a word w in the reduced lexicon is the "true" value of s (see Figure 44).

The Damerau-Levenshtein metric [11, 58] computes the distance between two strings as measured by the minimum cost sequence of "edit operations" — namely, deletions,

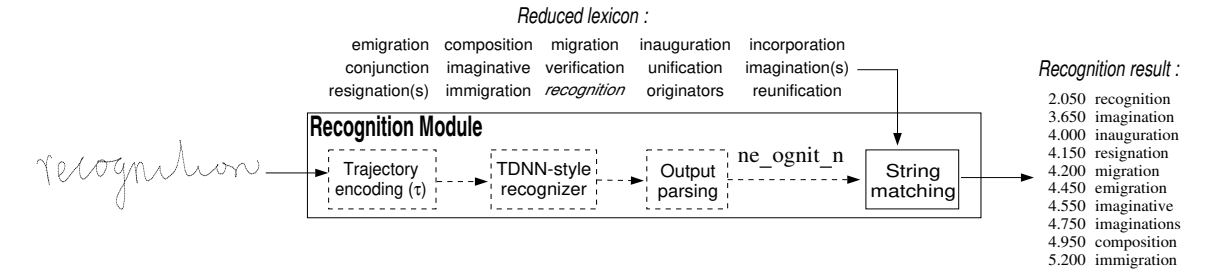


Figure 44: The role of string matching in the Recognition module: interpretation strings are matched with the reduced lexicon provided by the Filtering module to produce a final list of word choices (ranked by string distance score).

insertions, and substitutions — needed to change s into w. The term minimum edit distance was introduced by Wagner and Fischer, [97] who, simultaneously with Okuda *et al.* [72], proposed a dynamic-programming recurrence relation for computing the minimum-cost edit sequence.

Minimum edit distance techniques have been used to correct virtually all types of non-word mis-spellings, including typographic errors (e.g., mis-typing letters due to keyboard adjacency), spelling errors (e.g., doubling of consonants), and OCR errors (e.g., confusion of individual characters due to similarity in feature space). OCR-generated errors, however, do not follow the pattern of human errors [52]; specifically, a considerable number of the former are not one-to-one errors [43] but rather of the form $x_i \ldots x_{i+m-1} \rightarrow y_j \ldots y_{j+n-1}$ (where $m, n \geq 0$). This is particularly true in the script recognition domain, where ambiguities in letter segmentation and the presence of ligatures give rise to *splitting* (e.g., 'a'→'ci'), *merging* (e.g., 'cl'→'d') and *pair-substitution* errors (e.g., 'hi'→ 'lu').

Different extensions of the basic Damerau-Levenshtein metric are reported in the literature. Lowrance and Wagner [59] extend the metric to allow reversals of characters. Kashyap and Oommen [44] present a variant for computing the distance between a misspelled (noisy) string and every entry in the lexicon "simultaneously" under certain restrictions. Veronis [96] suggests a modification to compensate for phonographic errors (i.e., errors preserving pronunciation). In our word recognition problem we are concerned with the correction of errors due to improper character segmentation, on which relatively little work has been done. We are only aware of Bozinovic's attempt to model *merge* and *split* errors using a probabilistic finite state machine [6] (instead of through minimum edit distance methods). He, however, points out that the model "only approximates merging errors since it does not reflect what symbol the deleted one merged with".

Another explored direction of generalization to the Damerau-Levenshtein metric comes from assigning different weights to each operation as a function of the character or characters involved. Thus, for example, W_S('v'/'u') (the cost associated with the edit operation 'u' $\rightarrow$ 'v') could be smaller than W_S('v'/'q'). Tables of confusion probabilities modeling phonetic similarities, letter similarities, and mis-keying have been published. For a particular OCR device, this probability distribution can be estimated by feeding the device a sample of text and tabulating the resulting error statistics. However, the need remains for a framework permitting the analysis of how the various types of errors should be treated. That is, how does the cost of each operation relate to those of the others? Should $W_S(u/v)$ be less or greater than $W_M(z/xy)$ (the cost associated with the edit

operation $xy \rightarrow z$) for any characters u, v, x, y and z?

In a previously published paper [86] I have addressed these two issues, namely, (i) extending the basic Damerau-Levenshtein method to allow merges, splits and pair-substitutions, and (ii) developing a rationale for weight cost assignment to the operations. The main ideas are presented next.

6.6.1 Extension of the Damerau-Levenshtein metric

Let A be a finite set of symbols (the alphabet), and A^* be the set of strings over A. Let λ denote the null or empty string. Let $X = x_1 x_2 \cdots x_n$ and $Y = y_1 y_2 \cdots y_m$ be any two strings in A^* (we can assume Y to be a noisy version of X). Let $\alpha, \beta \in A^*$, $1 \leq p \leq n$, $1 \leq q \leq m$. Consider all possible ways of transforming Y into X. Now suppose that the i^{th} edit-sequence consisted of:

1. $\#S_i$ SUBSTITUTE operations of the form $y_q \rightarrow x_p$ (where $X = \alpha x_p \beta$ and $Y = \alpha y_q \beta$);
2. $\#D_i$ DELETE operations of the form $y_q \rightarrow \lambda$ (where $X = \alpha\beta$ and $Y = \alpha y_q \beta$);
3. $\#I_i$ INSERT operations of the form $\lambda \rightarrow x_p$ (where $X = \alpha x_p \beta$ and $Y = \alpha\beta$);

The Damerau-Levenshtein metric (DLM) computes a similarity value between X and Y as follows:

$$DLM(X/Y) = DLM(Y \rightarrow X) = \min_i(\#S_i \times W_S + \#D_i \times W_D + \#I_i \times W_I) \quad (1)$$

where W_S, W_D, and W_I are the non-negative costs associated with the corresponding operations. We use the notation $DLM(X/Y)$ instead of $DLM(X,Y)$ to emphasize that the problem is not symmetrical in general. Computing the DLM measure can be formulated as an optimization problem and a Dynamic Programming technique can be applied. The computation is carried out using the following recurrence relation [97, 72]:

$$d_{i,j} = \min \begin{cases} d_{i-1,j-1} + W_S(x_i/y_j) \\ d_{i,j-1} + W_D(y_j) \\ d_{i-1,j} + W_I(x_i) \end{cases} \quad (2)$$

with the base cases $d_{0,0} = 0$, $d_{i,0} = \sum_{k=1}^{i} W_D(y_k)$, and $d_{0,j} = \sum_{k=1}^{j} W_I(x_k)$. The value of $DLM(X/Y)$ is then given by $d_{m,n}$. The algorithm requires time proportional to the product of the lengths of the two strings (i.e., $O(nm)$), which is not prohibitive. Shortcuts must be devised when comparing the corrupted string with every word in a large lexicon [44]. Here, however, we are assuming the size of the lexicon is small because it is the result of the filtering process.

Although the above three edit operations (henceforth termed SUBSTITUTE, DELETE, and INSERT) have a strong data-transmission "flavor", since they were originally motivated by applications such as automatic detection and correction of errors in computer networks, they also suit the type of errors introduced by OCR's and other automatic reading devices. Insertions are needed to compensate for characters in the input which did not exceed a minimal recognition threshold; deletions are needed to get rid of false-positive responses — ligatures are a large source of false-positives in the script recognition

domain — and substitutions are needed to compensate for likely character confusions. The types of errors that these operations correct can be considered "recognition" errors. A different type of error occurs when adjacent characters are merged or split due to improper character "segmentation". To capture the fact that the merging of two characters into a third is not quite the same phenomenon as a substitution plus a deletion, we explicitly introduce the MERGE and SPLIT operations (note that in fact, a substitution can itself be modeled as a deletion plus an insertion). In cursive handwriting, for instance, the sequence 'ci' can easily be merged into an 'a'. This cannot be modeled meaningfully by a (context-free) substitution of 'c' for 'a' and a parallel deletion of 'i' (see Figure 45). The recurrence relation in Equation (2) can be easily extended to cope with merging and splitting errors by adding the minimization terms:

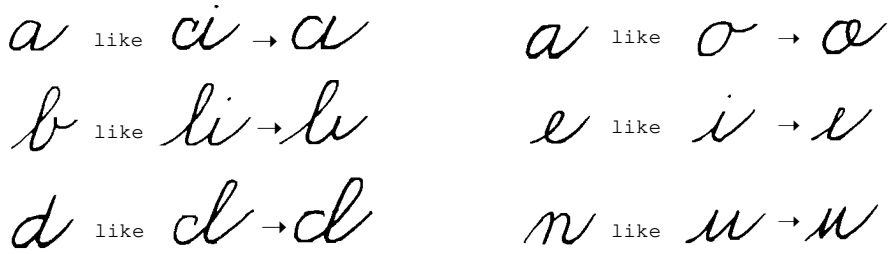
$$d_{i-2,j-1} \quad + \quad W_{\mathsf{merge}}(x_{i-1}x_i/y_j)$$

$$d_{i-1,j-2} \quad + \quad W_{\mathsf{split}}(x_i/y_{j-1}y_j)$$

We introduce here another edit operation that we term PAIR-SUBSTITUTE, to model a different type of phenomenon. MERGE and SPLIT model segmentation errors on the part of the recognizer — specifically, errors of omission and insertion of segmentation points. A third kind of segmentation error occurs when there is a "movement" of the segmentation point (e.g.,'mn'→'nm'). PAIR-SUBSTITUTE models this by substituting a pair of characters by another pair. The following term is thus also added to Equation (2):

$$d_{i-2,j-2} \quad + \quad W_{\mathsf{pair\text{-}substitute}}(x_{i-1}x_i/y_{j-1}y_j)$$

Just as a MERGE cannot be replaced by a substitution-plus-deletion, so a PAIR-SUBSTITUTE operation cannot be replaced by two parallel, but non-conjoined substitutions. All three operations can be thought of as capturing (a limited amount of) context-sensitivity, and so cannot be reduced to any set of "simpler" operations.



(a) (b)

Figure 45: Examples of common "look-alikes" occurring in cursive handwriting: (a) merge errors; these errors cannot be modeled meaningfully by a (context-free) substitution and a parallel deletion, and (b) substitution errors.

With the addition of the three new operations, we are faced with a compelling need to develop a rationale for any decisions we make about the relative costs associated with the operations. It turns out that stroke descriptions, developed in the context of the Filtering module, provide valuable information in comparing a test string (the string generated by the recognizer) and a reference string (an entry in the reduced lexicon). This idea was exploited to develop a framework for modeling the various types of errors, including segmentation ones, where operations were refined into categories according to the effect they have on the visual form of words. A set of recognizer-independent constraints that reflect the severity of the information lost due to each operation were identified; the resulting inequalities were solved to assign specific costs to the operations.

Table 1 presents the final cost assignment; further details about how these weights were derived are given in Appendix B.

VLS	PS	LS	LM	VLD	LD	LF	LI	US	UM	UD	UF	UI
0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	1.0	1.05	1.1	1.15	1.2

Table 1: Cost assignment for the refined set of operations: The prefixes 'VL', 'L' and 'U' to denote 'Very Likely', 'Likely', and 'Unlikely' operations, respectively. The letters S, D, I, and M refer to SUBSTITUTE, DELETE, INSERT, and MERGE respectively. F denotes a SPLIT – or fracture – and P denotes a PAIR-SUBSTITUTE.

Having determined costs for the set of refined operations, the next step was to decide which character or characters will be involved in each type of operation. These decisions were initially made by simply looking at the stroke description of each character. Such an assignment could then be refined based on secondary criteria such as closure of loops. For instance, one could decide not to categorize 'y' $\rightarrow$ 'q' as a VLS despite the similarity in the stroke descriptions for the two characters. Error statistics, when reliable, can and should also be incorporated into individual decisions about membership in the various categories of operations. Tables 2, 3, and 4 show the current letter assignment.

The superiority of the extended DLM over the traditional Damerau-Levenshtein metric (i.e., $W_S = W_D = W_I = 1$) was systematically evaluated in [86].

6.7 Testing of Recognition Module

The most common success measure used to determine the effectiveness of a recognition system is top-n accuracy, which is the probability with which the correct word appears

x_i	VLS	LS	x_i	VLS	LS	x_i	VLS	LS
a	u	o	j	—	—	s	—	l
b	—	l	k	—	h	t	—	—
c	—	—	l	—	—	u	v,n,a	o
d	—	l	m	w	n	v	u	—
e	i	—	n	u	v	w	m	u, v
f	—	—	o	—	a, u	x	—	—
g	y	j, q	p	—	—	y	g	j, q
h	—	k, l	q	g	y	z	—	—
i	e	r	r	—	i			

Table 2: The SUBSTITUTE table: Each entry in the table specifies the category (and cost) of that particular SUBSTITUTE. *eg*: 'l' $\in$ SUBSTITUTE('h', LS) means that W_S('h'/'l') = LS.

SPLIT $\downarrow$	a	b	d	h	k	u	u	w	w	y
MERGE $\uparrow$	ci	li	cl	li	lc	ee	ii	ev	iv	ij

Table 3: The SPLIT/MERGE table: The entries in the table are bi-directional. (This is not necessarily the case in every domain.) Going from the top row to the bottom row corresponds to a SPLIT, while the reverse corresponds to a MERGE.

in the first n entries of the ranked list of word choices output by the system. It is also important that systems are able to recognize the handwriting of multiple writers: writer-dependent tests quantify the system's ability to recognize handwriting styles already seen by the system during its training session; writer-independent tests are, on the other hand, intended to measure the walk-up recognition abilities of the system (i.e., the recognition accuracy that may be expected by a writer unseen by the system without having to go through a training session).

We report performance with a vocabulary of 21,000 words. For training data, we

{he, hi}	⇔	lu	{em, im}	⇔	un
{me, mi}	⇔	nu	{ey, iy}	⇔	uj
{ue, ui}	⇔	{eu, iu}	{ew, iw}	⇔	uv
{mn, mu}	⇔	nm			
{hn, hu}	⇔	lm	{bj, hj}	⇔	ly
hv	⇔	lw			
mv	⇔	nw	mj	⇔	ny
wv	⇔	uw	wj	⇔	uy

Table 4: Valid PAIR-SUBSTITUTE possibilities: Very few PAIR-SUBSTITUTE's are plausible; like SPLIT/MERGE's, PAIR-SUBSTITUTE's tend to be bi-directional also.

used 2,443 lowercase cursive word images (11,691 characters) from 55 different writers. There were 516 different words in this data set. Tables 5 and 6 describe the data used for testing of the Recognition module and summarize performance results. The detailed characteristics of the data used for evaluation of this module are explained in Appendix C.

Data Set			**Word Level Accuracy**		
Images	*Words*	*Writers*	*Top 1*	*Top 5*	*Top 10*
443	50	20	91.6%	97.9%	99.3%

Table 5: Writer-dependent Test.

In 6 cases, out of the 443 images used in the writer-dependent test, the reduced lexicon was "adjusted" to include the truth.

In 52 cases, out of the 466 images used in the writer-independent test, the reduced lexicon was "adjusted" to include the truth.

Data Set			Word Level Accuracy		
Images	*Words*	*Writers*	*Top 1*	*Top 5*	*Top 10*
466	300	9	62.4%	82.4%	88.1%

Table 6: Writer-independent Test.

6.8 Discussion of Recognition Module

We have described a complete system for the recognition of on-line cursive handwriting which has been tested on a moderately large database of words. The system establishes a new syntactic approach to efficiently deal with large lexicon sizes, exploits the underlying dynamics of cursive handwriting generation by means of a simple "novel" representation scheme of the pen trajectory and processing of delayed-strokes, successfully applies the method of time-delay neural networks, and demonstrates how to customize a string matching function to achieve higher error correction rates.

The recognition performance of the system can be improved in a number of ways. For instance, a scheme for combining the recognition scores resulting from the peak parsing process and the distance values returned by the string matching routine is desirable; currently, scores from the peak parsing stage are being discarded. Enhancements in the normalization stage, however, appear to be the most important gain source: a badly estimated median letter height could result in a trajectory were letters are unusually long (or short), therefore impeding proper recognition. Alternative normalization schemes can also be explored; one could, for example, force all "strokes" to be of the same length (i.e., represented by the same number of points). The difficulty here would be to come up

with a suitable definition of stroke; furthermore, such a scheme could result in different parts of a letter being represented with different resolution.

Experimentals results showed that the system has good writer-independent capabilities; a simple writer-adaptation mechanism can, however, be provided by means of the string similarity function: edit-operation costs formally derived could be automatically "tune" to more accurately compensate for the type of errors a given writer is prone to commit.

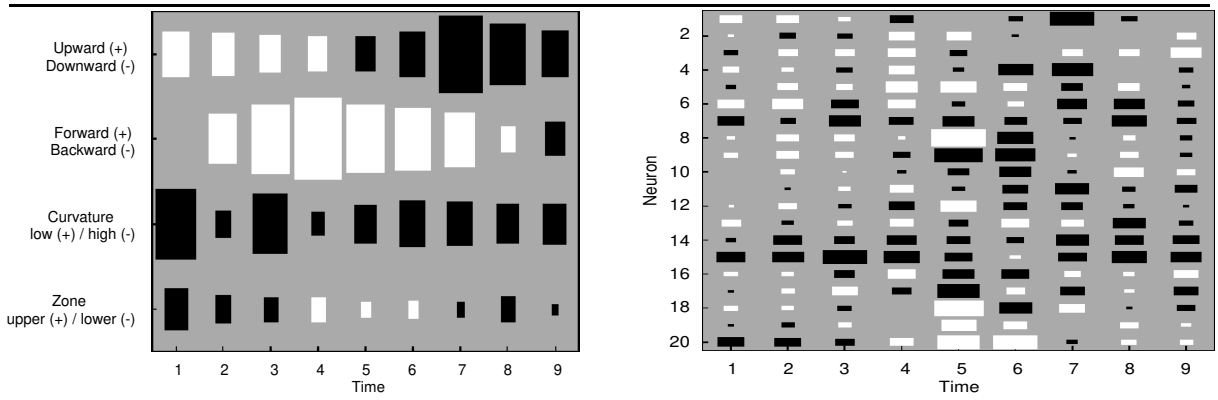
Our network differs from Waibel's phoneme recognition TDNN [98] in that we do not perform external integration of the activity of the output units over time. Waibel's network was built to sum the squares of the activations of multiple output unit copies which could each see a different portion of the input pattern. Using this replicated output unit architecture, the network did not require supervision in the time domain during training, that is, the target information did not include information about where the patterns occurred, only about whether a particular pattern occurred. This training strategy reduces considerably the effort needed to prepare a training data set because segmentation labels become unnecessary. However, it also makes the amount of data required for proper training significantly larger. Any pair of images will have a large number of patterns in common if no information is supply as to the location of the common patterns; in order for the network to figure out which of these patterns are the intended ones, a very large number of examples would have to be shown to it. While it is difficult to obtain precise segmentation information for a set of recorded utterances,

it is easy to identify intercharacter boundaries in images of cursive script. The lack of positional information in the target signal would also make the peak identification process more difficult since the notions of expected peak size, expected peak width and expected inter-peak gap would no longer exist.

Finally, more interesting than the network's performance is the fact that the network managed to learn meaningful weight patterns from the training data. The rectangular patterns on Figure 46 show some of the weights that the network developed. Weights are plotted as a grid of squares: each square's area represents a weights magnitude and each square's color represents a weights sign; black for negative weights, white for positive. Time is represented by the horizontal axis of the weight matrix, and input activation from the layer to which the weight is connected by the vertical axis.

Figure 46(a) shows the weight kernel corresponding to one of the 15 units in the first hidden layer; the input to this unit is a temporal window of size 9 in the input trajectory. It is easy to determine that this unit is acting as a "cusp" detector: the white squares at the top of the first four frames of the weight pattern show that the pen is moving upwards (e.g., $\nearrow$); it then moves downwards (e.g., $\searrow$) for the next five frames. The white squares in the second row indicates a forward pen movement (e.g., $\rightarrow$) and the black squares in the third row specify a region of high curvature. All the weights in the fourth row have small magnitudes, indicating that the "zone" parameter is relatively irrelevant for this feature.

Figure 46(b) shows the weights that the network developed for transmitting activation



(a) (b)

Figure 46: Examples of weight kernels: (a) weights associated with a "cusp" detecting unit in the first hidden layer, and (b) weights learned to connect the output unit for the letter 'e' with the second hidden layer.

from the second hidden layer to the output unit corresponding to the letter 'e'. Because the largest squares are at, and near, frame 5, it shows that the network has effectively learned to focus its attention on the center of the input receptive field; this is important because a small letter like 'e' has a short temporal representation (on average, a letter 'e' occupies only 25 frames out of the 96 the input window of the network holds) and so the network must learn to "ignore" extra or unnecessary input.

Chapter 7

Conclusions

A hierarchical model for large vocabulary recognition of on-line handwritten cursive words, motivated by several psychological research findings about the human perception of handwriting, has been developed and tested. The model is composed of two modules that operate as two independent classifiers based on semi-orthogonal sources of information; the first one tentatively recognizes words, the second one performs letter-analytical tests. In particular, the following issues were explored:

- efficiently dealing with large reference dictionary sizes;

  It was demonstrated that that the visual configuration of a word written in cursive script can be captured by a stroke description string. The stroke description scheme identifies 9 different types of strokes, some of which capture spatio-temporal information such as retrograde motion of the pen. This idea was used to good effect in a lexicon-filtering module that operates on lexicon sizes of over 20,000 words.

- the role of dynamic information over traditional feature-analysis models in the recognition process;

It was empirically demonstrated that the dynamic pattern of pen motion in cursive handwriting carries enough information for recognition. The approach has the advantage of effectively avoiding the problem of touching or overlapping characters.

- the incorporation of letter context and avoidance of error-prone segmentation of the script by means of the scanning window concept;

  A neural network-based recognizer was successfully trained to recognize what is centered in its input window as it slides along a character string, effectively avoiding the need for an explicit character segmentation step. The network receptive field was designed so as to capture a limited amount of context-sensitivity, and in this way account for the co-articulation phenomena that makes cursive handwriting recognition a difficult task.

- the use of domain-specific information in the string-to-string similarity computation;

  The Damerau-Levenshtein string difference metric was generalized in two ways to more accurately compensate for the types of errors that are present in the script recognition domain. First, the basic dynamic programming method for computing such a measure was extended to allow for merges, splits and two-letter substitutions. Second, edit operations were refined into categories according to the effect

they have on the visual "appearance" of words.

Experimental results clearly showed that an on-line handwritten word recognition (HWR) system designed according to these ideas can be successful.

Appendix A
Production Rules for Syntactic Matching

In this appendix we list the production rules used in the process of deriving English words from a string of stroke primitives which extracted from a given word image.

One symbol substituted by one letter:

$$
\begin{array}{lcl}
A & \rightarrow & b|d|f|h|k|l|t \\
B & \rightarrow & b|d|f|g|h|i|j|k|l|p|q|t|y|z \\
D & \rightarrow & f|g|j|p|q|y|z \\
M & \rightarrow & a|c|e|i|m|n|o|r|s|u|v|w|x \\
R & \rightarrow & a|c|o \\
U & \rightarrow & a|b|...|z \\
C & \rightarrow & r \\
L & \rightarrow & s
\end{array}
$$

Two symbols substituted by one letter:

$$
\begin{array}{lcl}
AM & \rightarrow & b|h|k \\
AU & \rightarrow & b|h|k \\
BM & \rightarrow & b|h|f|k|p \\
BU & \rightarrow & b|h|f|k|p \\
DM & \rightarrow & p \\
DU & \rightarrow & p \\
DL & \rightarrow & p \\
MA & \rightarrow & d \\
UA & \rightarrow & d \\
RA & \rightarrow & d \\
MC & \rightarrow & o|r|u|v|w \\
MD & \rightarrow & g|q|y|z
\end{array}
$$

Two symbols substituted by one letter (cont.):

$$
\begin{array}{rcl}
UD & \rightarrow & g|q|y|z \\
MB & \rightarrow & d|g|q|y \\
MM & \rightarrow & a|m|n|o|r|u|v|w|x|z \\
MU & \rightarrow & a|d|g|m|n|o|q|r|u|v|w|y|x|z \\
MR & \rightarrow & x \\
RC & \rightarrow & o \\
RD & \rightarrow & g|q \\
RB & \rightarrow & d|g|q \\
RM & \rightarrow & a|o \\
RU & \rightarrow & a|d|g|q \\
UM & \rightarrow & a|b|f|h|k|m|n|o|p|r|u|v|w|x|z \\
UU & \rightarrow & a|b|d|g|h|k|m|n|o|p|q|r|u|v|w|x|y|z \\
CM & \rightarrow & r \\
CU & \rightarrow & r \\
AK & \rightarrow & k \\
UK & \rightarrow & k \\
AA & \rightarrow & k \\
AR & \rightarrow & k \\
UR & \rightarrow & k \\
BR & \rightarrow & k \\
AL & \rightarrow & b|k \\
UL & \rightarrow & b|k|p \\
BL & \rightarrow & k|p \\
UC & \rightarrow & b|o|r|u|v|w
\end{array}
$$

Three symbols substituted by one letter:

$$
\begin{array}{rcl}
UUM & \rightarrow & k|m|q|w \\
UMM & \rightarrow & k|m|w \\
UMU & \rightarrow & k|m|w \\
UUU & \rightarrow & k|m|w \\
MMM & \rightarrow & m|w \\
MMU & \rightarrow & m|w \\
MUU & \rightarrow & m|w \\
MUM & \rightarrow & m|q|w \\
AMM & \rightarrow & k \\
AMU & \rightarrow & k \\
AKM & \rightarrow & k \\
AKU & \rightarrow & k
\end{array}
$$

Three symbols substituted by one letter (cont.):

$$
\begin{array}{lcl}
AUM & \rightarrow & k \\
AUU & \rightarrow & k \\
BMM & \rightarrow & k \\
BMU & \rightarrow & k \\
BKM & \rightarrow & k \\
BKU & \rightarrow & k \\
BUM & \rightarrow & k \\
BUU & \rightarrow & k \\
UKU & \rightarrow & k \\
UKM & \rightarrow & k \\
MMC & \rightarrow & w \\
MUC & \rightarrow & w \\
UMC & \rightarrow & w \\
UUC & \rightarrow & w \\
RDM & \rightarrow & q \\
RUM & \rightarrow & q \\
MDM & \rightarrow & q \\
UDM & \rightarrow & q
\end{array}
$$

Appendix B
A Typology of Recognizer Errors

The relative cost of an edit operation is defined in terms of its effect on the visual "appearance" of words. More precisely, three primary criteria are outlined on the basis of which the six categories of operations — namely, SUBSTITUTE, DELETE, INSERT, MERGE, FRACTURE, and PAIR-SUBSTITUTE — are refined into Likely, Unlikely and Impossible operations (mnemonic: a SPLIT is a FRACTURE). A basic ordering of the refined set of operations is achieved based on these three criteria. These primary criteria, and other secondary criteria, are then applied to further restrict the legal cost-ranges for the various operations.

Based on the stroke description scheme developed in the context of the Filtering module, we define (in order of decreasing significance) three measures that help us judge the quantity and quality of the "damage" that a particular edit operation inflicts on the shape of a word:

(M1) the number and positions of prominent strokes

(M2) the number and positions of *all* strokes together

(M3) the number and positions of characters

The primary criteria are defined as the changes caused by the edit operation to M1, M2 and M3. Thus, a SUBSTITUTE would *a priori* be considered less damaging than a DELETE — a SUBSTITUTE would maintain all three variables at approximately the same value, whereas a DELETE would damage M2 and M3 at the very least. Similarly a FRACTURE would be rated as being less expensive than an INSERT, since a FRACTURE increases M3 alone, while an INSERT increases both M2 and M3.

B.1 Refining the operations

The six basic operations are refined using the measures M1, M2 and M3. We will use the prefixes 'VL', 'L' and 'U' to denote 'Very Likely', 'Likely', and 'Unlikely' operations, respectively. Table 7 summarizes the definitions.

The cases that are not covered by the definitions in the table are defined to be Impossible operations. For instance, a MERGE of the form 'pd' → 'q' simply cannot happen under any normal circumstances. The distinction between Unlikely and Impossible operations is that Unlikely operations could happen in very noisy situations or when there are serious generation problems, while Impossible operations cannot be conceived of even under such circumstances. The Impossible operations do not figure in any of our analysis, because their cost is set to ∞ in order to prevent them from being included in any minimum-cost edit sequence. We refer to the set {VLS, LS, US, PS, LM, UM, VLD, LD, UD, LF, UF, LI, UI} as the set of refined operations.

Basic operation	Refine-ment	Definition (based on change in word shape)	Measures affected	Example
SUBSTITUTE	VLS	No change in shape	None	'n' → 'u'
	LS	Prominent strokes static	M2	'b' → 'l'
	US	Prominent stroke(s) demoted	M1, M2	'h' → 'n'
PAIR-SUBST	PS	No change in shape	None	'hi' → 'lu'
MERGE	LM	No change in shape	M3	'ij' → 'y'
	UM	Prominent strokes static	M2, M3	'uj' → 'y'
DELETE	VLD	Single Median deletion	M2, M3	'r' → ' λ'
	LD	Prominent strokes static	M2, M3	'a' → ' λ'
	UD	Everything else	All three	'y' → ' λ'
FRACTURE (SPLIT)	LF	No change in shape	M3	'u' → 'ii'
	UF	Prominent strokes static	M2, M3	'd' → 'ch'
INSERT	LI	Single Median insertion	M2, M3	' λ' → 'i'
	UI	Everything else	All three	' λ' → 'q'

Table 7: **Refining the basic operations:** The prefixes 'VL', 'L' and 'U' qualify each operation as being 'Very Likely', 'Likely', and 'Unlikely', respectively. The cases not covered here are deemed Impossible.

The stroke description scheme, along with the measures M1, M2 and M3, provides a solid foundation for the refinement of the basic operations. However, assigning relative costs to this set of finer divisions among the operations is not a straightforward process. Nor is it easy to justify any intuitions about the ordering of the basic operations. Here, too, our three measures M1, M2, and M3 come to the rescue.

B.2 The basic ordering

In measuring the distance of the test string from the reference string, the system is in fact attempting to recover from (potential) recognizer errors. The nature of cursive

handwritten text is such that the recognizer is more liable to treat non-strokes (such as ligatures and connection strokes) as valid strokes, than to overlook valid strokes. Therefore, in this domain, a DELETE operation should be penalized less than an INSERT should. (The opposite may be true in the domain of machine-printed text.)

Given this, and on the basis of the three measures and their importance relative to each other, certain general conclusions about the various operations can be drawn. First, we note that every Unlikely operation must be penalized more than any Likely operation should. A glance at Table 7 will reveal that Unlikely operations tend to upset a superset of the measures affected by Likely operations. Indeed, most of the Unlikely operations directly affect the value of M1. We denote this "meta-ordering" by the formula:

$$\mathrm{L}\mathcal{X} < \mathrm{U}\mathcal{Y} \tag{3}$$

where the $\mathcal{X}$ and $\mathcal{Y}$ stand as place-holders for any of the six basic operations. Table 7 also confirms that the six operations can be grouped based on their effect on the three measures:

- SUBSTITUTE & PAIR-SUBSTITUTE preserve M3, while none of the others do; further, VLS preserves the recognizer's segmentation decisions while PS overrules them.

- MERGE & DELETE decrease M3; DELETE decreases M2 also.

  (MERGE only re-groups strokes, and so does not affect M2.)

- SPLIT & INSERT increase M3; INSERT increases M2 also.

(SPLIT, like MERGE, also re-groups strokes while preserving M2.)

Consequently, and because of the (domain-specific) assumption concerning DELETE's *vs.* INSERT's, we can conclude that:

$$\text{SUBSTITUTE} < \text{PAIR-SUBSTITUTE} < \text{MERGE} < \text{DELETE} < \text{SPLIT} < \text{INSERT} \quad (4)$$

Based on Inequalities (3) and (4) we can now refine the ordering as follows:

$$\text{VLS} < \text{PS} < \text{LS} < \text{LM} < \text{VLD} < \text{LD} < \text{LF} < \text{LI}$$

$$< \text{US} < \text{UM} < \text{UD} < \text{UF} < \text{UI}$$

We refer to this as the basic ordering of the set of refined operations. As noted above, PS gets placed after VLS because the former does not preserve segmentation information. PS gets positioned before LS because, unlike LS, it preserves M2.

B.3 Additional constraints

Secondary constraints are now applied to further restrict the ranges of costs that the various operations can be assigned. These secondary constraints are specified in an attempt to model phenomena that are not captured by the basic ordering above. The first pair of such constraints are:

$$\text{LM} < \text{VLS} + \text{VLD} \quad (5)$$

$$\text{LF} < \text{VLS} + \text{LI} \quad (6)$$

These constraints capture the fact that a MERGE (such as 'cl' → 'd') is not just a DELETE plus a SUBSTITUTE, and therefore should not be penalized to the same extent. Although these constraints are trivially satisfied by the basic ordering, they help to illustrate the types of relationships that the additional constraints try to capture. Next, we specify:

$$\text{LD} \quad < \quad 2 \times \text{VLD} \tag{7}$$

This is motivated by the fact that an LD "corresponds" to two VLD's (in that a VLD deletes a single median stroke, while an LD deletes two), but damages M3 (the character count) less than a pair of VLD's. We then add two complementary constraints as follows:

$$\text{LM} + \text{LI} \quad < \quad \text{US} + \text{VLS} \tag{8}$$

$$\text{LF} + \text{VLD} \quad < \quad \text{US} + \text{VLS} \tag{9}$$

These constraints model the idea that an edit sequence where a Likely MERGE was coupled with a Likely INSERT would be preferred to a sequence where an Unlikely SUBSTITUTE was forced. Thus, for example, in comparing the test string 'suli' with the reference string 'such', we would prefer the edit sequence [' λ' → 'c'; 'li' → 'h'] over the sequence ['l' → 'c'; 'i' → 'h'].

B.4 Solving for the cost ranges

In order to find values for the costs that must be associated with the various edit operations, we solve the set of inequalities formed by adding the additional constraints to the

basic ordering. We used the simplex method of linear programming to solve the set of inequalities. The objective function that we maximized was:

$$(\mathrm{US} - \mathrm{LI}) \;+\; {}^{-}(\mathrm{UI} - \mathrm{US}) \;+\; {}^{-}(\mathrm{LI} - \mathrm{VLS})$$

Each of the three terms in the objective function captures a specific aspect of the structure of the set of refined operations. Maximizing the first term $(\mathrm{US} - \mathrm{LI})$ corresponds to stating that the Unlikely operations should be placed as far from the Likely operations as possible. Maximizing ${}^{-}(\mathrm{UI} - \mathrm{US})$ is the same as minimizing $(\mathrm{UI} - \mathrm{US})$ and so states that the Unlikely operations should be grouped together. Similarly, the third term ${}^{-}(\mathrm{LI} - \mathrm{VLS})$ implies that the Likely operations should also be clustered.

This objective function increases monotonically with the Unlikely operations, and so we need to specify an upper bound for the costs of the various operations. Therefore, we bound the operations from above by adding the constraint $\mathrm{UI} \leq 1.5$. We further specify that $\mathrm{US} = 1.0$ based on the reasoning that this corresponds to the "traditional" notion of SUBSTITUTE. Additionally, we set the step-size for the costs to be 0.05 and specify a minimum cost of 0.2 for any operation. These numbers constitute reasonable estimates, and are certainly open to refinement based on feedback through performance figures. With these constraints, we obtained the cost assignment in Table 8.

VLS	PS	LS	LM	VLD	LD	LF	LI	US	UM	UD	UF	UI
0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	1.0	1.05	1.1	1.15	1.2

Table 8: **Cost assignment for the refined set of operations.**

Appendix C
Experimental Data

A carefully designed body of data plays an important role in the construction of successful pattern recognition systems. Furthermore, the extent to which experimental results are meaningful is closely related to the degree in which the data set chosen accurately models the occurrences of data in the task addressed.

In this appendix I describe the cursive handwriting corpus used for training and evaluation of the recognition system presented in the preceding chapters.

C.1 Desirable Corpus Characteristics

Two major characteristics are highly desirable in a data corpus intended for building a pattern recognition system [45]: (i) it must contain enough examples of each class so that regularities can be learned, and (ii) it must allow for a meaningful evaluation of the system (i.e., the conditions under which the data is collected and the amount of variability present in the data are similar to those in which the system will be used).

Kassel [45] enumerates the following factors as major sources of variability in hand-

writing data:

- variety in hardware platforms used to record the data;
- spontaneity of the writing

  i.e., are subjects instructed to write in a particular style or in a way that is natural to them;
- unit of writing

  i.e., are samples of individual letters, strings of characters, or full sentences and paragraphs being collected;
- allograph variation

  i.e., within a particular writing style many symbol styles or allographs are possible;
- letter case;
- subject's gender, age, and hand favored for writing;
- experimental conditions

  i.e., are lines or boxes provided, is visual or acoustic prompting used, what writing surface and stylus is used.

Clearly, collection of a large amount of data is required in order to capture all this sources of handwriting variability. One is often, however, limited by time and resource constraints. A trade off must thus be made between variability covered and effort devoted to data collection, keeping in mind the intended application.

The data used in my experiments is the result of three collection efforts carried out at CEDAR during the past two years. I will refer to these three data sets as "First25", "Second25", and "Sentence". Table 9 summarizes how each of these data sets conform to the different variability criteria. All three data sets were collected using a Wacom model SD-311 opaque tablet connected to a SUN workstation. This device uses a cordless inking stylus with a "natural" feel (i.e., not bulky) and has an electrostatic surface to hold in position paper placed on top of it .

Data Set	*Spontaneity of Writing*	*Unit of Writing*	*Allographic Variation*	*Letter Case*
First25	limited	words	allowed	lower only
Second25	limited	words	allowed	lower only
Sentence	no restrictions	sentences	allowed	mostly lower

Data	*Subject's*			*Exp. Condition*		
Set	*Gender*	*Age*	*Hand*	*Boxed*	*Baseline*	*Prompting*
First25	both	20-30	right	no	no	visual
Second25	both	17-35	right	no	no	visual
Sentence	both	15-50	both	no	no	aural

Table 9: Variability factors covered by our handwriting corpus.

C.2 The First25 Data Set

Because the recognizer developed along the course of this research was intended for cursive handwriting, my initial goal was to collect samples of cursive words that would provide a roughly uniform number of occurrences for each of the lowercase English letters. I believe that when feasible data size is limited, it is more important to have enough samples of

every class at the expense of distorting the natural distribution (i.e., the letter frequency in the English language). Additionally, because subjects were volunteers, and not paid, a small number of words that would not require more than 45 minutes or so to collect was needed.

I estimated that around 75 words could be written and stored in this amount of time. To more easily observe regularities in the data, I decided that the same set of words should be written multiple times. I thus selected a set of 25 different words and asked donors to write them at least three times. To meet the frequency requirement, a simple algorithm for selecting the 25 words from a 60,000 entry dictionary was implemented; the algorithm randomly selects 25 words, tests letter coverage, and if necessary replaces the word with more occurrences of the highest frequency (max_freq) letter in the set with a new randomly selected word containing the lowest frequency letter (min_freq). The stopping condition was formally specified by: $max_freq \leq n \times min_freq$. Figure 47 lists the final set of 25 words found with $n = 3.6$ and the corresponding letter distribution (in the entire 60,000 word dictionary we have $n \approx 75$).

The letters 't' and 'x' were intentionally not included in the word set, for at the time these samples were collected I wanted to avoid dealing with delayed strokes. Ten persons (mostly graduate students at CEDAR) volunteered to write the words; they were instructed to write cursive but no constraint was imposed on size, slant, or orientation. The resulting data is summarized in Table 10.

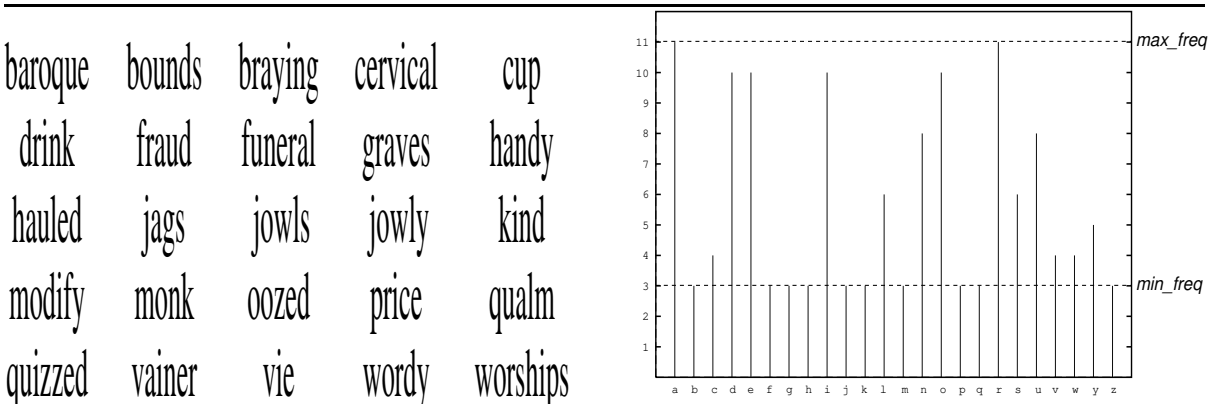


Figure 47: Words in the First25 data set: (a) words to be used for data collection to achieve a roughly uniform number of letter samples, and (b) corresponding letter frequencies.

Images	*Words*	*Writers*	*Characters*
825	25	10	4521

Table 10: The First25 data set.

C.3 The Second25 Data Set

In designing the Second25 word data set, I was interested in, as well as letter coverage, letter pairs. Because our character recognizer was being designed to include a notion of letter context, it appeared relevant to cover common letter pairs. For this purpose, the frequency of occurrence of all possible letter pairs in a 21,000 word dictionary was computed; the top ranked letter pairs in this lexicon are shown in Table 11. It should be noticed, however, that frequency count alone is ineffective in revealing letter pairs that are meaningful (e.g., 'qu') but which occur infrequently. More sophisticated measures are needed to detect these pairs [45].

Rank	Pair	Rank	Pair	Rank	Pair	Rank	Pair	Rank	Pair
1	in √	7	ti	13	st	19	ri √	25	io √
2	er √	8	ng √	14	ar √	20	or √	26	it
3	re √	9	te	15	le √	21	de	27	ro
4	on √	10	en √	16	ra	22	li √	28	ne
5	ed	11	an √	17	al √	23	co	29	ic √
6	es √	12	at	18	nt	24	is	30	se √

Table 11: Common data pairs: from a 21,000 word lexicon as ranked by pair frequency.

Twenty different sets of 25 words each, were generated with the algorithm described in the previous section. The set covering the largest number of the data pairs listed in Table 11 was selected. Figure 48 shows the words included in the selected set which found with $n = 3.5$; data pairs covered by the set are indicated with a √ in Table 11.

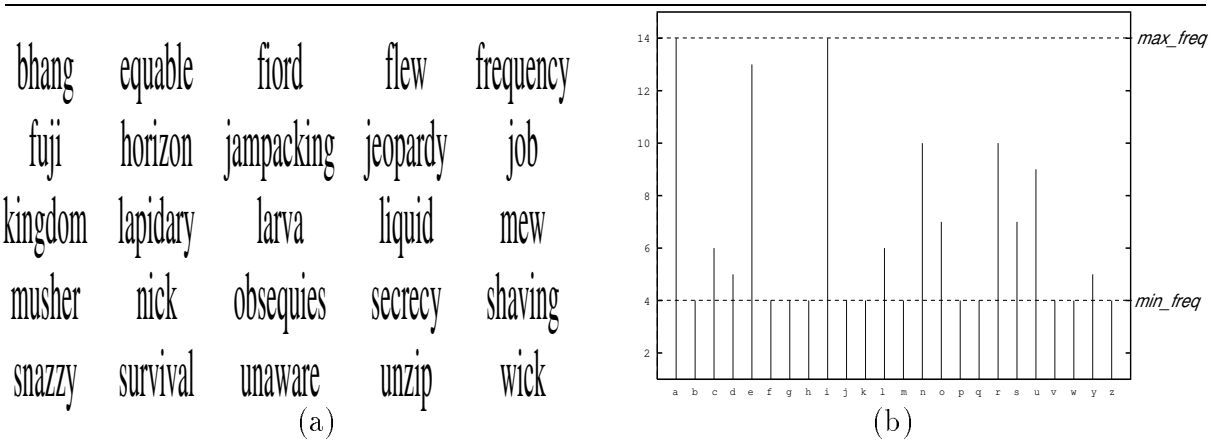


Figure 48: Words in the Second25 data set: (a) words to be used for data collection to achieve a roughly uniform number of letter samples and coverage of common letter pairs, and (b) corresponding letter frequencies.

A new pool of 10 volunteers were instructed to write these words in cursive style. The resulting data is summarized in Table 12.

Images	*Words*	*Writers*	*Characters*
750	25	10	4620

Table 12: The Second25 data set.

C.4 The Sentence Data Set

The last component of our experimental data is a small subset of a large database more recently collected at CEDAR and that became possible thanks to an external grant from the Linguistic Data Consortium. The entire database contains slightly over 100,000 words including alphabetic and numeric data, and collected under a variety of conditions (e.g., using combs, boxes, and nothing at all). About half of this data was collected in a very unconstrained manner: donors were asked to write freely passages which presented to them aurally and on a sentence by sentence basis. It is a commonly-held belief that aural prompting, as opposed to visual prompting, avoids influencing handwriting style and size. Furthermore, by having donors write full phrases, as opposed to isolated words, the resulting handwriting data characteristics will resemble more those to be encountered by the recognizer when deployed in a general text recognition application.

A total of twelve passages, selected from a variety of different genres of text, were used; two male speakers recorded the corresponding phrases, which were digitized to permit playback at will. Each writer wrote three to four passages, containing approximately 15 sentences/phrases each. A program we developed played the sentences of the selected passage on a pair of headphones; after each sentence had being played, it prompted the

writer to write the sentence on the tablet. Progress e was controlled by the subject through three main on-screen buttons: "Play Sentence" to play the current sentence, "Read Tablet" to activate recording of pen coordinates, and "Save Sentence" to save to a file the recorded handwriting. Very little supervision was given to subjects but a "host", who received them into the lab, was always available for assistance.

Subjects were recruited from both the SUNY at Buffalo community and the general population through posters and ads in the school newspaper. A brief competency test in writing and listening English was given to them but they were not required to be native speakers. Modest compensation was provided in return for their participation. Subjects were asked to fill out a short biographic information questionnaire; this questionnaire included an entry for indicating his or her writing style (i.e., cursive, printed, or mixed) which was determined by the lab host based on visual inspection of the sheet(s) of paper the subject had written on. About a third of the data was labeled as cursive.

Sentence data was semi-automatically segmented into words; individual words were then transcribed using a graphical interface specifically written for this task. Recorded data was displayed on the screen and the prompt text was supplied as a default string to be edited by the transcriber. At this time the "style" label of each word, which inherited from the label given to the corresponding writer, was updated if necessary. Character level truth was generated for 5609 words which labeled as cursive. Four undergraduate students assisted on this task using a tool I developed (see Figure 49); the tool provided them with a special cursor to mark points in the images corresponding to "reasonable"

begin and end points for each letter. Truthers were instructed to edit the ASCII truth of words when necessary, and to use a special character ('?') when, in their judgment, a letter in the image was so poorly written that it was difficult to make any sense of it. This was necessary because we found multiple instances where letter elements were missing, sometimes due to careless writing and sometimes as a result of inaccuracies in the pen-up/pen-down indication, presumably because of the faster speed at which people write full sentences as opposed to isolated words. An example of this situation is illustrated in Figure 49.

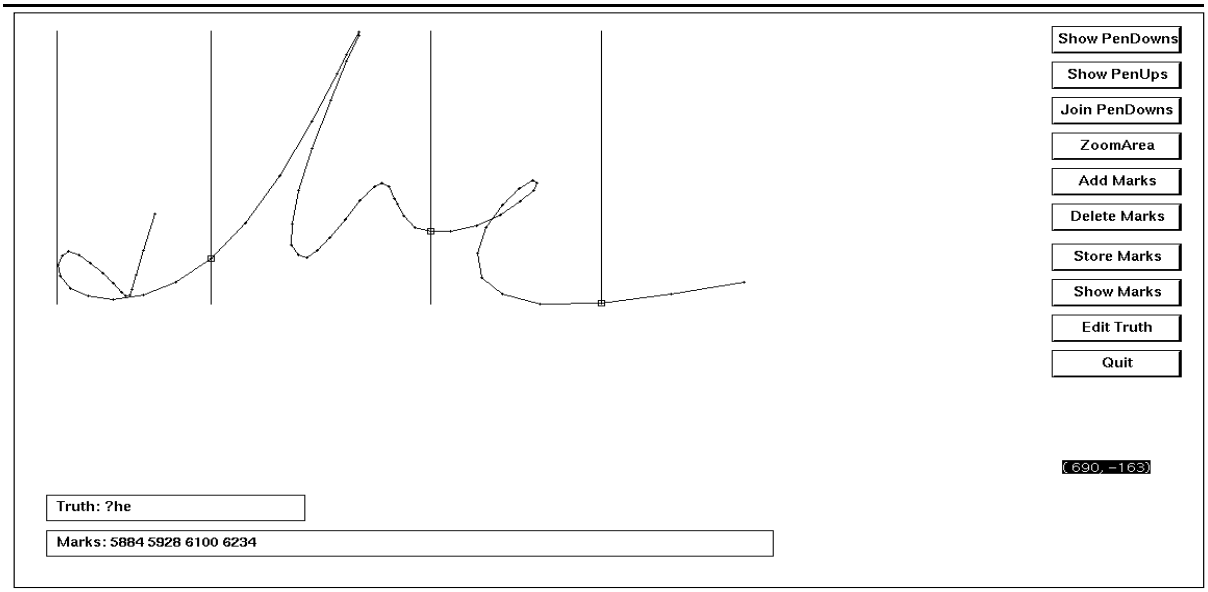


Figure 49: Example of data truthing screen for cursive words: points corresponding to inter-character boundaries are marked with a vertical cursor. In this example, the ASCII truth is updated from 'the' to '?he' because the first character is judged illegible.

Images of the truthed cursive words were shuffled and assigned by an impartial party to one of 3 sets: setI (2907) intended for training, setII (959) intended for development testing, and setIII (1743) intended for acceptance testing. Normally, one test set is

sufficient to assess recognition performance on "unseen" data. But, during development, a system can become tailored to a particular data set used for evaluation by the various tweaks and corrections made. Consequently, a second test set is commonly set aside to be used only for final acceptance. In my experiments, I used data from setI for evaluation of the Filtering Module and training of the Recognition module; data from SetII for testing of the Recognition Module (only one time), and never used SetIII.

To evaluate the performance of the Filtering module images from SetI that did not contained capital letters nor the special character '?' were combined with those of the First25 and Second25 data sets. The table presented in Figure 50(a) summarizes the resulting data set. Prior to training of the Recognition module, this data set had to be further "cleaned" to remove images that contained errors in truthing, or where the preprocessing operations had badly failed. It was possible to automatically detect these two problems by computing for every character, the means (μ_{cl}) and variances (σ_{cl}) of their lenght (i.e., the number of points between the begin and end marks) across a subset of 300 words for which the truth was visually inspected. Then, images containing characters with lenght not within $\mu_{cl} \pm 3.5\sigma_{cl}$ were simply discarded. The resulting data set was split into a training and writer-dependent test set for the Recognition module; the tables presented in Figure 50(b) summarize them.

A separate, writer-independent, test set for the Recognition module was obtained from setII after applying the same cleaning operations described above. The resulting data set is summarized in Table 13; examples of images in this set that were properly

Data Set	*Images*	*Words*	*Writers*	*Characters*
First25	825	25	10	4521
Second25	750	25	10	4620
SetI	2111	534	37	10712
Total	3686	584	57	19853

(a)

Writer-Dependent Test Data Set			
Images	*Words*	*Writers*	*Characters*
443	50	20	2987

Training Data Set			
Images	*Words*	*Writers*	*Characters*
2443	516	55	11691

(b)

Figure 50: The amount of data available in our handwriting corpus: (a) data used for evaluation of the Filtering module, and (b) how this data was split into a training and test set for training and (writer-dependent) evaluation of the Recognition module.

recognized are shown in Figure 51.

Writer-Independent Test Data Set			
Images	*Words*	*Writers*	*Characters*
466	300	9	2453

Table 13: Test data: the amount of data used for writer-independent evaluation of the Recognition module.

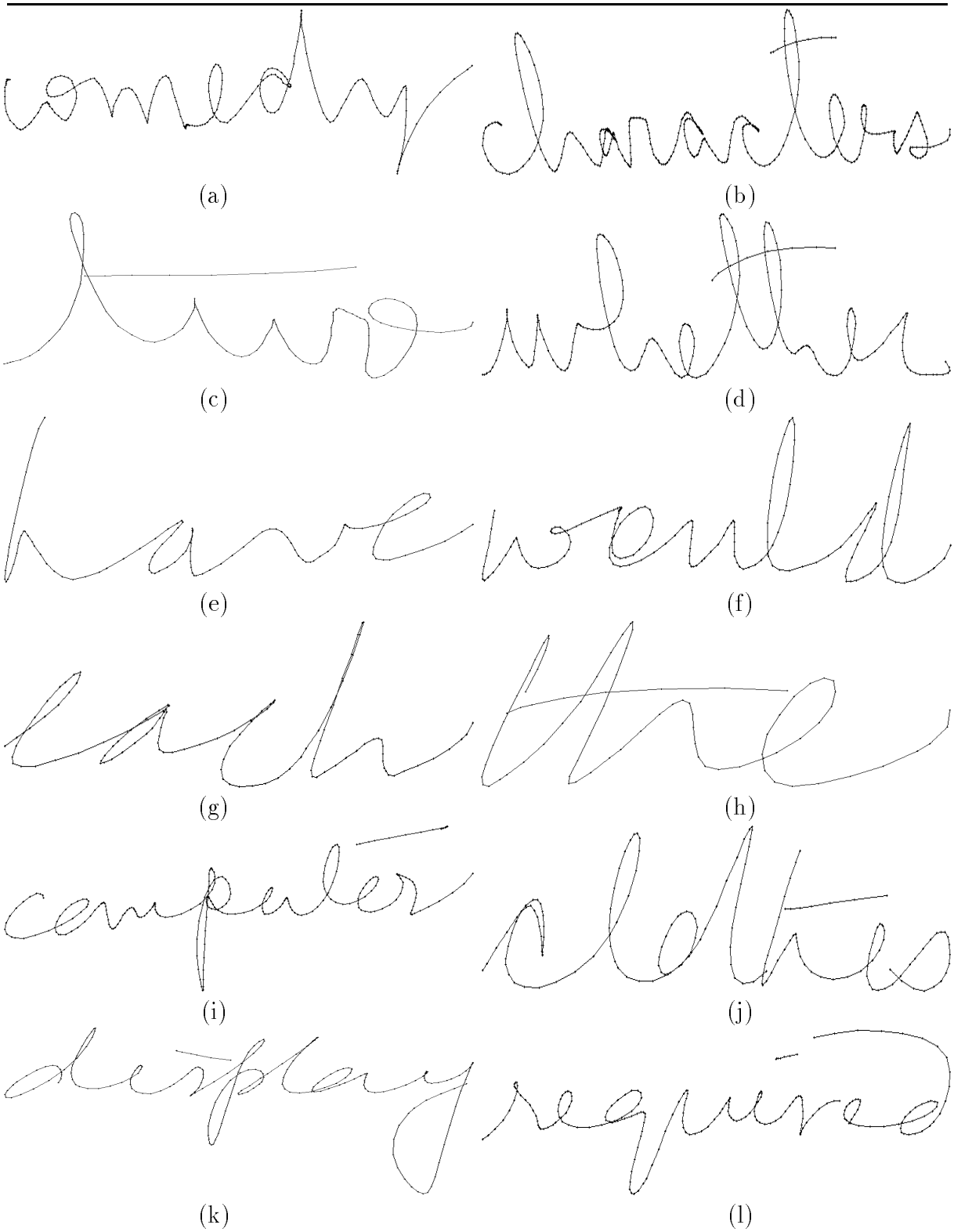


Figure 51: Test image examples: (a) 'comedy', (b) characters, (c) two, (d) whether, (e) 'have', (f) 'would', (g) 'each', (h) 'the', (i) 'computer', (j) 'clothes', (k) 'display', (l) 'required'.

References

[1] J.A. Anderson and E. Rosenfeld. *Neurocomputing: Foundations of Research.* MIT Press, 1988.

[2] M.K. Babcock and J.J. Freyd. Perception of dynamic information in static handwritten forms. *American Journal of Psychology*, 101(1):111–130, 1988.

[3] D.H. Ballard and C.M. Brown. *Computer Vision.* Prentice-Hall, 1982.

[4] Y. Bengio. A connectionist approach to speech recognition. *Intl. Jour. Pattern Recog. Artif. Intell.*, 7(4):3–22, 1993.

[5] H. Bouma. Visual recognition of isolated lower-case letters. *Vision Research*, 11:459–474, 1971.

[6] R. Bozinovic and S.N. Srihari. A string correction algorithm for cursive script recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:655–663, 1982.

[7] E.R. Brocklehurst and P.D. Kenward. *Preprocessing for cursive script recognition.* NPL Report DITC 132/88, 1988.

[8] M.K. Brown and S. Ganapathy. Cursive script recognition. In *Intl. Conference on Cybernetics and Society*, pages 47–51, 1980.

[9] M.K. Brown and S. Ganapathy. Preprocessing techniques for cursive script word recognition. *Pattern Recognition*, 16:447–458, 1983.

[10] D.W.J. Corcoran and R.O. Rouse. An aspect of perceptual organization involved in reading typed and handwritten words. *Quarterly Journal of Experimental Psychology*, 22:526–530, 1970.

[11] F.J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[12] J. Dayhoff. *Neural network architectures.* Van Nostrand Reinhold, 1990.

[13] G. Dimauro, S. Impedovo, and G. Pirlo. A stroke-oriented approach to signature verification. In S. Impevodo and J.C. Simon, editors, *From Pixels to Features III: Frontiers in Handwriting Recognition.* Elsevier Science Publishers, 1992.

[14] R.O. Duda and P.E. Hart. *Pattern classification and scene analysis.* John Wiley & Sons, New York, 1973.

[15] L.D. Earnest. Machine recognition of cursive script. *Information processing 1962 (Proc. IFIP Congr.)*, pages 462–466, 1962.

[16] S. Edelman, T. Flash, and S. Ullman. Reading cursive handwriting by alignment of letter prototypes. *International Journal of Computer Vision*, 5(3):303–331, 1990.

[17] R.W. Ehrich and K.J. Koehler. Experiments in the contextual recognition of cursive script. *IEEE Transactions on Computers*, 24:182–194, 1975.

[18] D.L. Elliott. A better activation function for artificial neural networks. Technical Report TR93-8, Institute for Systems Research, University of Maryland, 1993.

[19] S.E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-DD-88-162, Computer Science Department, Carnegie Mellon University, 1988.

[20] R.F.H. Farag. Word level recognition of cursive script. *IEEE transactions on Computers*, 28:172–175, 1979.

[21] J.T. Favata. *Recognition of Cursive, Discrete and Mixed Handwritten Words Using Character, Lexical and Spatial Constraints.* PhD thesis, State University of New York at Buffalo, 1992.

[22] N.S. Flann and S. Shekhar. Recognizing on-line cursive handwriting using a mixture of cooperating pyramid-style neural networks. In *World Congress on Neural Networks*, Oregon, 1993.

[23] D.M. Ford. *On-line recognition of connected handwriting.* PhD thesis, University of Nottingham, 1991.

[24] H. Freeman. Computer processing of line-drawing images. *Computing Surveys*, 6:57–97, 1974.

[25] J.J. Freyd. Representing the dynamics of a static form. *Memory & Cognition*, 11(4):342–346, 1983.

[26] L.S. Frishkopf and L.D. Harmon. Machine reading of cursive script. *4th London Symposium on Information Theory*, pages 300–316, 1961.

[27] K.S. Fu. *Syntactic Pattern Recognition Applications*. Springer-Verlag, 1977.

[28] T. Fujisaki, H.S.M. Beigi, C.C. Tappert, M. Ukelson, and C.G. Wolf. Online recognition of unconstrained handprinting: a stroke-based system and its evaluation. In S. Impevodo and J.C. Simon, editors, *From Pixels to Features III: Frontiers in Handwriting Recognition*. Elsevier Science Publishers, 1992.

[29] T. Fujisaki, T.E. Chefalas, J. Kim, C.C. Tappert, and C.G. Wolf. On-line run-on character recognizer: design and performance. *Journal of Pattern Recognition and Artificial Intelligence*, 1:123–136, 1991.

[30] S. Geva and J. Sitte. A constructive method for multivariate function approximation by multilayer perceptrons. *IEEE Transactions on Neural Networks*, 23(4):621–624, 1992.

[31] R.C. Gonzalez and M.G. Thomason. *Syntactic Pattern Recognition*. Addison-Wesley, 1978.

[32] W. Guerfali and R. Plamondon. Normalizing and restoring on-line handwriting. *Pattern Recognition*, 26(3):419–431, 1993.

[33] I. Guyon, P. Albrecht, Y. LeCun, J. Denker, and W. Hubbard. Design of a neural network character recognizer for a touch terminal. *Pattern Recognition*, 24(2):105–119, 1991.

[34] I. Guyon, D. Henderson, P. Albrecht, Y. LeCun, and J. Denker. Writer independent and writer adaptive neural network for on-line character recognition. In S. Impevodo and J.C. Simon, editors, *From Pixels to Features III: Frontiers in Handwriting Recognition*. Elsevier Science Publishers, 1992.

[35] N.Z. Hakim, J.J. Kaufman, G. Cerf, and H.E. Meadows. Cursive script online character recognition with a recurrent neural network model. In *International Joint Conference on Neural Networks*. IEEE, 1992.

[36] R. Hecht-Nielsen. *Neurocomputing*. Addison-Wesley, 1973.

[37] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley, 1991.

[38] C.A. Higgins and R. Whitrow. On-line cursive script recognition. In *Human Compuer Interaction - INTERACT 84*. IFIP, Elsevier Science Publishers, 1985.

[39] J. Hoffman, J. Skrzypek, and J.J. Vidal. Cluster network for recognition of handwritten cursive script characters. *Neural Networks*, 6:69–78, 1993.

[40] J.M. Hollerbach. An oscillation theory of handwriting. *Biological Cybernetics*, 39:139–156, 1981.

[41] K. Hornik, M. Stinchcombe, and H. White. Multi-layer feed forward networks are universal approximators. *Neural Networks*, 2:359–368, 1989.

[42] W.Y. Huang and R.P. Lippman. Comparisons between neural net and conventional classifiers. In *First IEEE Conference on Neural Networks*, San Diego, 1987.

[43] M.A. Jones, G.A. Story, and B.W. Ballard. Integrating multiple knowledge sources in a Bayesian OCR post-processor. In *ICDAR-91*, pages 925–933, St. Malo, France, 1991.

[44] R.L. Kashyap and B.J. Oommen. An effective algorithm for string correction using generalized edit distances. *Information Sciences*, 23:123–142, 1981.

[45] R.H. Kassel. *A Comparison of Approaches to On-Line Handwritten Character Recognition.* PhD thesis, Massachusetts Institute of Technology, 1995.

[46] J.D. Keeler, D.E. Rumelhart, and W. Leow. Handwritten digit recognition with a backpropagation network. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems II.* Morgan Kaufmann, 1990.

[47] J.D. Keeler, D.E. Rumelhart, and W. Leow. Integrated segmentation and recognition of hand-printed numerals. In R.P. Lippman, J.E. Moody, and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems III.* Morgan Kaufmann, 1991.

[48] D.D. Kerrick and A.C. Bocik. Microprocessor-based recognition of handprinted characters from a tablet input. *Pattern Recognition*, 21(5):525–537, 1988.

[49] D.E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison Wesley, 1973.

[50] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, New York, second edition, 1988.

[51] H. Kojima and T. Toida. On-line hand-drawn line-figure recognition and its application. In *9th International Conference on Pattern Recognition*, Rome, Italy, 1988.

[52] K. Kukich. Automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[53] K.J. Lang, A.H. Waibel, and G.E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:23–43, 1990.

[54] A. Lapedes and R. Farber. How neural nets work. In D.Z. Anderson, editor, *Neural Information Processing Systems*, pages 442–456. 1988.

[55] Y. LeCun. Generalization and network design strategies. In R. Pfeifer, F. Fogelman-Soulie, and L. Steels, editors, *Connectionism in Perspective.* Elsevier Science Publishers, 1989.

[56] D.S. Lee and S.N. Srihari. Dynamic classifier combination using neural networks. In *SPIE/IS&T Conference on Document Recognition*, San Jose, CA, 1995.

[57] C.G. Leedham, A.C. Downton, C.P. Brooks, and A.F. Newell. On-line acquisition of pitman's handwritten shorthand as a means of rapid data entry. In *Human Computer Interaction - INTERACT 84*. IFIP, Elsevier Science Publishers, 1985.

[58] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics-Doklady*, 10(8):707–710, 1966.

[59] R. Lowrance and R.A. Wagner. An extension of the string-to-string correction problem. *Journal of the ACM*, 23(2):177–183, 1975.

[60] F.L. Maarse, R.G.J. Meulenbroek, H.L. Teulings, and A. Thomasen. Computational measures for ballisticity handwriting. In R. Plamondon, C.Y. Suen, J.G. deschenes, and G. Poulin, editors, *Proceedings of the Third International Symposium on Handwriting and Computer Applications*. 1987.

[61] G.L. Martin. *Using a neural network to recognize hand-drawn symbols*. MCC Technical Report ACT-HI-232-90, 1990.

[62] G.L. Martin and J.A. Pittman. Recognizing hand-printed letters and digits using backpropagation learning. *Neural Computation*, 3:258–267, 1991.

[63] G.L. Martin and M. Rashid. Recognizing overlapping hand-printed characters by centered object integrated segmentation and recognition. In R.P. Lippman, J.E.

Moody, and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems IV*. Morgan Kaufmann, 1992.

[64] G.L. Martin, M. Rashid, and J.A. Pittman. *Integrated segmentation and recognition through exhaustive scans or learned saccadic jumps.* MCC Technical Report NN-175-92, 1992.

[65] W.S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943. Reprinted in Anderson and Rosenfeld, 1988.

[66] P. Mermelstein and M. Eden. Experiments on computer recognition of connected handwriting words. *Information and Control*, 7:255–270, 1964.

[67] M.L. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[68] P. Morasso. Neural models for cursive script handwriting. In *IEEE Intl. Conference on Neural Networks*, volume 2, pages 539–542, 1989.

[69] P. Morasso, L. Barberis, S. Pagliano, and D. Vergano. Recognition experiments of cursive dynamic handwriting with self-organizing networks. *Pattern Recognition*, 26(3):451–460, 1993.

[70] P. Morasso and F.A. Mussa Ivaldi. Trajectory formation and handwriting: A computational model. *Biological Cybernetics*, 45:131–142, 1982.

[71] K. Ohmori. On-line handwritten kanji character recognition using hypothesis generation in the space of hierarchical knowledge. In *Third International Workshop on Frontiers in Handwriting Recognition (IWFHR III)*, 1993.

[72] T. Okuda, E. Tanaka, and T. Kasai. A method for the correction of garbled words based on the Levenshtein metric. *IEEE Transactions on Computers*, 25(2):172–178, 1976.

[73] J.A. Pittman. Recognizing handwritten text. MCC Report, 1992.

[74] R. Plamondon and F.J. Maarse. An evaluation of motor models of handwriting. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1060–1072, 1989.

[75] R. Plamondon, P.Yergeau, and J.J. Brault. A multi-level signature verification system. In S. Impevodo and J.C. Simon, editors, *From Pixels to Features III: Frontiers in Handwriting Recognition*. Elsevier Science Publishers, 1992.

[76] Y. Qiao and C.G. Leedham. Segmentation and recognition of handwritten pitman's shorthand outlines using an interactive heuristic search. *Pattern Recognition*, 26(3):433–441, 1993.

[77] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of The IEEE*, 77(2):257–285, 1989.

[78] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, Washington, DC, 1962.

[79] D.E. Rumelhart. Theory to practice: A case study - recognizing cursive handwriting. In *Third NEC Symposium Computational Learning and Cognition*, Princeton, New Jersey, 1992.

[80] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning internal representations by error propagation*, volume 1, pages 318–362. Bradford Books, 1986.

[81] M. Schenkel, H. Weissman, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line hand-printed words. In *Advances in Neural Information Processing Systems V*. Morgan Kaufmann, 1993.

[82] L. Schomaker. Using stroke or character-based self-organizing maps in the recognition of on-line, connected cursive script. *Pattern Recognition*, 26(3):443–450, 1993.

[83] J. Schuermann. Pattern classification: a unified view on statistical and neural approaches. Manuscript for publication, 1994.

[84] T.J. Sejnowski and C.R. Rosemberg. *NETtalk: a parallel network that learns to read aloud*. JHU EECS Technical Report JHU/EECS-86/01, 1986.

[85] G. Seni and E. Cohen. External word segmentation of off-line handwritten text lines. *Pattern Recognition*, 27(1):41–52, 1994.

[86] G. Seni, V. Kripåsundar, and R.K. Srihari. Generalizing edit distance for handwritten text recognition. In *SPIE/IS&T Conference on Document Recognition*, San Jose, CA, 1995.

[87] J.C. Simon and O. Baret. Cursive word recognition. In *From Pixels to Features II*. Elsevier Science Publishers, 1992.

[88] Y. Singer and N. Tishby. A discrete dynamical approach to cursive handwriting analysis. Technical Report CS93-4, Institute of Computer Science, The Hebrew University of Jerusalem, 1993.

[89] J. Skrzypek and J. Hoffman. Visual recognition of script characters and neural network architectures. In E. Gelembe, editor, *Neural Networks: Advances and Applications*. Elsevier Science Publishers, 1991.

[90] P. Smolensky. *Neural and conceptual interpretation of PDP models*, volume 2. MIT Press, 1986.

[91] R.K. Srihari and C.M. Baltus. Incorporating syntactic constraints in recognizing handwritten sentences. In *International Joint Conference on Artificial Intelligence (IJCAI-93)*, Chambery, France, 1993.

[92] M. Stinchcombe and H. White. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *IJCNN*, pages 613–617, 1989.

[93] C.C. Tappert. Adaptive on-line handwriting recognition. In *7th International Conference on Pattern Recognition*, Montreal, Canada, 1984.

[94] C.C. Tappert. Speed, accuracy, and flexibility trade-offs in on-line character recognition. *Intl. Journal of Pattern Recognition and Artificial Intelligence*, 5:79–95, 1991.

[95] C.C. Tappert, C.Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:787–808, 1990.

[96] J. Veronis. Computerized correction of phonographic errors. *Computers and the Humanities*, 22:43–56, 1988.

[97] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[98] A.H. Waibel, T.Hanazawa, G.E. Hinton, K.Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 37:328–339, 1989.

[99] P.J. Werbos. *Beyond regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[100] B. Widrow and M.E. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record*, 4:94–104, 1960. Reprinted in Anderson and Rosenfeld, 1988.

[101] I. Yoshimura and M. Yoshimura. On-line signature verification incorporating the direction of pen movement. In S. Impevodo and J.C. Simon, editors, *From Pixels*

to Features III: Frontiers in Handwriting Recognition. Elsevier Science Publishers, 1992.

[102] A. Zimmer. Do we see what makes our script characteristic or do we only feel it ? modes of sensory control in handwriting. *Psychological Research*, 44:165–174, 1982.